ЛАБОРАТОРНА РОБОТА 2

Тема: Вивчення кросплатформеності Java програм на прикладі алгоритмів сортування

Теоретична частина

Алгоритми сортування є класичними прикладами теорії алгоритмів, за якими вивчають складність алгоритмів і обчислювальну ефективність вирішення однієї задачі.

Загальна постановка задачі сортування числових даних

Вхід: послідовність з n чисел $(a_1,a_2,\ldots, a_n)$.
Вихід: перестановка (зміна слідування) $(a_1',a_2',\ldots,a_n')$ вхідної послідовності таким чином, щоб в результаті стало виконуватись, наприклад, відношення не зменшення

$$a_1' \le a_2' \le \ldots \le a_n'$$

Сортування вставкою

Це один із самих простих у реалізації алгоритмів сортування. Він має наступні властивості:

- достатньо ефективний на малих наборах даних;
- ефективний у тих випадках, коли до відсортованих даних додається новий елемент;
- це стійкий алгоритм сортування, тому що не змінює послідовність елементів, які вже відсортовано;
- може сортувати потокові дані по мері їх отримання;
- не потребує тимчасової пам’яті, наприклад, для стеку.

Ідея алгоритму полягає в тому, що послідовно один за одним беруться елементи із вхідного одномірного масиву даних і вставляються на потрібну позицію у вже відсортованій частині. Ця процедура повторюється до тих пір, поки набір вхідних даних не вичерпається. Пошук потрібної позиції відбувається шляхом послідовного порівняння елементу сортування з елементами відсортованої частини, починаючи з найбільшого елементу в ній у бік зменшення індексу.

Приклад сортування методом простої вставки на мові Java

Створіть файл з ім’ям *insertion.java* та додайте в нього наведений нижче код на Java.
Для компіляції виконайте у командному рядку

javac *insertion.java*

Результатом повинно бути утворення файлу *insertion.class.* Для запуску програми після компіляції виконайте в командному рядку

java *insertion 10000*

де 10000 — кількість елементів в масиві, які отримують випадкові значення та будуть відсортовані.

```
public class insertion {

// метод doInsertion виконує сортування
// перших n елементів масиву a[]

      public static void doInsertion(double a[], int n){
            double temp;
            int j;
            for(int i=1; i<n; i++){
                  temp=a[i];
                  j=i-1;
                  while((j>=0)&&(temp<a[j])){
                        a[j+1]=a[j];
                        j--;
                  }
                  a[j+1]=temp;
            }
//  для контролю виконання сортування треба прибрати символи коментаря //
//          for(int i=0;i<n;i++){
//                System.out.println("a["+i+"]="+a[i]);
//       }
      }

      public static void main(String[] args) {

            double[] a=new double[1000000];
// кількість елементів задаємо у командному рядку
    int n=Integer.parseInt(args[0]);
// задаємо випадкові значення для елементів
            for(int i=0; i<n; i++){
                  a[i]=1000.0*Math.random()/(Math.random()+Math.random());
            }
            long t1, t2;
// визначаємо системний час у мілісекундах (з 01.01.1970) перед сортуванням
            t1=System.currentTimeMillis();
            doInsertion(a, n);
// визначаємо системний час після сортуванням
            t2=System.currentTimeMillis();
// обчислюємо час виконання сортування
            System.out.println("time = "+(double)(t2-t1)/1000+" s");
      }
```

Пряме упорядкування

Метод прямого упорядкування вимагає напряму вирішити дану задачу. Тому спочатку необхідно знайти найменший за значенням елемент масиву. Його буде перенесено до позиції з індексом 1. В іншій частині масиву, починаючи з індексу 2, знову виконується пошук найменшого за значенням елементу і його буде перенесено до позиції 2 і т.д.

Приклад сортування прямим упорядкуванням на мові Java

Створіть файл з ім’ям ***directsort.java*** та додайте в нього наведений нижче код на Java. Для компіляції виконайте в командному рядку

javac *directsort.java*

Результатом повинно бути утворення файлу ***directsort.class.*** Для запуску цієї програми виконайте у командному рядку

java *directsort* 10000

```
public class directsort {

// метод doSort дозволяє виконати сортування
// перших n елементів масиву a[] методом прямого упорядкування

  public static void doSort(double a[], int n){
    double temp;
    for(int i=0; i<n-1; i++){
      for(int j=i+1; j<n; j++){
       if(a[i]>a[j]){
       temp=a[i];
       a[i]=a[j];
       a[j]=temp;
       }
     }
   }
//  для контролю виконання сортування необхідно прибрати символи коментаря
//    for(int i=0;i<n;i++){
//      System.out.println("a["+i+"]="+a[i]);
//       }
  }

  public static void main(String[] args) {

    double[] a=new double[1000000];
// кількість елементів задається у командному рядку
    int n=Integer.parseInt(args[0]);
// задаємо випадкові значення для елементів
    for(int i=0; i<n; i++){
      a[i]=1000.0*Math.random()/(Math.random()+Math.random());
    }
    long t1, t2;
// визначаємо системний час в мілісекундах перед сортуванням
    t1=System.currentTimeMillis();
    doSort(a, n);
// визначаємо системний час в мілісекундах після сортування
    t2=System.currentTimeMillis();
// обчислюємо час виконання сортування
    System.out.println("time = "+(double)(t2-t1)/1000+" s");
  }

}
```

Метод обміну («бульбашки»)

Метод полягає в тому, що, починаючи з першого елементу масиву і до передостанього виконують послідовне порівняння двох сусідніх елементів. Якщо елемент з меншим індексом має менше значення, то елементи залишаються на тих самих місцях. Якщо елемент з меншим індексом має

більше значення, ніж сусідній елемент з більшим індексом, то елементи міняють місцями. Дія повторюється послідовно для всіх елементів. Якщо при черговому проході по масиву виконується хоч би одна перестановка, то прохід по масиву повторюється. Повторення проходів робиться до тих пір, поки не буде проходу по масиву без перестановок. Такий масив буде відсортованим.

Приклад сортування методом обміну на мові Java

Створіть файл з ім'ям ***blebsort.java*** і додайте до нього код програми на Java.
Для компіляції виконайте у командному рядку

javac *blebsort.java*

Для запуску програми з командного рядку виконайте

java *blebsort* 10000

```
public class blebsort{
// метод doSort виконує сортування
// перших n елементів масиву a[] методом обміну
  public static void doSort(double a[], int n){
    double temp;
    boolean flag=true;
    while(flag){
    flag=false;
    for(int i=0; i<n-1; i++){
      if(a[i]>a[i+1])
      {
       temp=a[i];
       a[i]=a[i+1];
       a[i+1]=temp;
       flag=true;
      }
     }
    }
//  для контролю виконання сортування необхідно прибрати символи коментаря
//    for(int i=0;i<n;i++){
//      System.out.println("a["+i+"]="+a[i]);
//       }
  }

  public static void main(String[] args) {
    double[] a=new double[1000000];
//  кількість елементів задаємо у командному рядку
    int n=Integer.parseInt(args[0]);
// задаємо випадкові значення для n елементів
    for(int i=0; i<n; i++){
      a[i]=1000.0*Math.random()/(Math.random()+Math.random());
    }
    long t1, t2;
// визначаємо системний час у мілісекундах перед сортуванням
    t1=System.currentTimeMillis();
    doSort(a, n);
// визначаємо системний час у мілісекундах після сортування
    t2=System.currentTimeMillis();
// обчислюємо час виконання сортування
    System.out.println("time = "+(double)(t2-t1)/1000+" s");
  }
}
```

Метод швидкого сортування

За цим методом на першому етапі сортування елементів масиву $a_1,a_2,...,a_n$ обирається такий елемент, значення якого використовується в якості опорного. Відносно цього елементу проводиться упорядкування усіх інших елементів масиву. Елементи масиву переставляються так, щоб для певного індексу j всі переставлені елементи $a_1,a_2,...,a_j$ мали значення менші за значення опорного елементу, а всі елементи $a_{j+1},a_{j+2},...,a_n$ — значення, більші або такі самі, що і опорного елементу.
Процедуру повторюють рекурсивно для множини елементів $a_1,a_2,...,a_j$ та $a_{j+1},a_{j+2},...,a_n$ задля упорядкування цих множин окремо. Тому що значення усіх елементів у першій множені меншу за значення елементів другої множини, то вихідний масив буде відсортовано правильно.

Приклад програми qsort на мові Java

Створіть файл с ім'ям ***myqsort.java*** і додайте до нього програмний код на Java, який наведено нижче. Для компіляції виконайте у командному рядку

javac *myqsort.java*

Для запуску після компіляції виконайте у командному рядку

java *myqsort* 10000

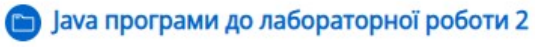
```
public class myqsort {
// Сортування перших n елементів масиву a[] методом швидкого сортування
    public static void doSort(double a[], int lb, int rb){
    double temp;
        if((lb+1==rb)&&(a[lb]>a[rb])){
          temp=a[lb];
          a[lb]=a[rb];
          a[rb]=temp;
        }
        if(lb+1<rb){
         int j=(lb+rb)/2;
         double key=a[j];
          a[j]=a[lb];
          a[lb]=key;
         int i_left=lb+1;
         int i_right=rb;

        while(i_left<i_right){
          while((a[i_left]<key)&&(i_left<i_right)) i_left++;
          while((key<=a[i_right])&&(i_left<=i_right)) i_right--;
          if(i_left<i_right){
            temp=a[i_left];
            a[i_left]=a[i_right];
            a[i_right]=temp;
          }
        }
           if(i_left>=i_right){
          temp=a[i_right];
          a[i_right]=a[lb];
          a[lb]=temp;
        }
        doSort(a,lb,i_right-1);
        doSort(a,i_right+1,rb);
      }
    }
    public static void qSort(double a[], int n){
          doSort(a,0,n-1);
//  для контролю сортування приберіть символи коментаря
```

```
//       for(int i=0;i<n;i++) System.out.println("a["+i+"]="+a[i]);
      }
    public static void main(String[] args) {
      double[] a=new double[10000000];
//  кількість елементів, які будуть сортуватись у масиві
//  задаємо у командному рядку
      int n=Integer.parseInt(args[0]);
// задаємо випадкові значення для цих елементів
      for(int i=0; i<n; i++){
        a[i]=1000.0*Math.random()/(Math.random()+Math.random());
      }
      long t1, t2;
// визначаємо системний час у мілісекундах перед сортуванням
      t1=System.currentTimeMillis();
      qSort(a, n);
// визначаємо системний час у мілісекундах після сортування
      t2=System.currentTimeMillis();
// обчислюємо час виконання сортування
      System.out.println("time = "+(double)(t2-t1)/1000+" s");
    }
  }
```

ПРАКТИЧНЕ ЗАВДАННЯ

1. Для виконання лабораторної роботи повинно бути два набори відкомпільованих програм на Java. Один набір створюється на основі наведених вище прикладів реалізацій алгоритмів, а другий знаходиться у відповідній теці курсу:



Програми, що знаходяться в теці відкомпільовано у версії Java SE 7 ще у 2013 році.

2. Необхідно визначити ефективність сортування кожної із програм. Для цього кожна із програм запускається у командному рядку наступним чином:

java *програма_сортування кількість_елементів*

Значення *кількість_елементів* змінюється від 10000 до 100000 з кроком у 10000. Тобто для кожної програми повинно бути отримано 10 значень часу виконання програми. Для програм **qsort** значення *кількість_елементів* слід змінювати від 100000 до 1000000 з кроком у 100000.

3. Для кожного типу реалізації програми (старої та нової компіляції) необхідно побудувати графіки залежності часу виконання від кількості елементів, що сортуються.

4. Порівняйте отримані залежності часу виконання та зробіть висновки.