SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

Algoritmi i strukture podataka

Vježba 3.

Sorting

Ivan Banovac, 220

Datum izvođenja vježbe:
01.12.2025

Zadatak 1. – Selection Sort
U ovom zadatku implementira se rekurzivni Selection Sort algoritam za sortiranje niza cijelih brojeva. Algoritam pronalazi najmanji element u nizu i postavlja ga na odgovarajuće mjesto, a zatim se isti postupak ponavlja za ostatak niza.

Selection.cs:
```
{
    class Selection
    {
        public static void Sort(int[] array, int start)
        {
            if (start >= array.Length - 1)
                return;
            int min = start;
            for (int i = start + 1; i < array.Length; i++)
            {
                if (array[i] < array[minIndex])
                {
                    min = i;
                }
            }
            int temp = array[start];
            array[start] = array[min];
            array[min] = temp;
            Sort(array, start + 1);
        }
    }
```

Program.cs:
```
using System;
namespace Selection
    class Program
    {
        static void Main(string[] args)
        {
            int[] array = { 9, 8, 4, 11, 5, 7, 3, 1 };
            Console.WriteLine("Originalni niz:");
            foreach (var num in array) Console.Write(num + " ");
            Console.WriteLine();
            Selection.Sort(array, 0);
            Console.WriteLine("Sortirani niz:");
            foreach (var num in array) Console.Write(num + " ");
            Console.WriteLine();
        }
    }
}
```

Rezultat:

```
Originalni niz:
9 8 4 11 5 7 3 1
Sortirani niz:
1 3 4 5 7 8 9 11
```

Glavna metoda *Sort* prima niz cijelih brojeva i indeks od kojeg sortiranje počinje. Algoritam funkcionira tako da u dijelu niza u kojem se trenutno nalazi, pronađe najmanji element i zamijeni ga s elementom na početku niza, a zatim rekurzivno poziva sam sebe za ostatak niza,

te povećavajući indeks za 1. Ukoliko je startni indeks jednak posljednjem indeksu u nizu, rekurzija se zaustavlja. U glavnom programu inicijaliziramo zadani niz brojeva {9, 8, 4, 11, 5, 7, 3, 1} te pozivamo metodu sortiranja i ispisujemo rezultat.

Zadatak 2. – Quick Sort

U ovom zadatku implementira se rekurzivni Quick Sort algoritam koristeći delegate. Sortiranje se provodi nad nizom objekata tipa Student, prvo po imenu, a zatim po prosječnoj ocjeni.

Student.cs:

```
using System;
namespace QuickSort
{
    class Student
    {
        private string name;
        private double grade;
        public Student(string name, double grade)
        {
            this.name = name;
            this.grade = grade;
        }
        public override string ToString()
        {
            return name +  ":" + grade;
        }
        public static bool CompareName(object a, object b)
        {
            Student s1 = (Student)a;
            Student s2 = (Student)b;
            return string.Compare(s1.name, s2.name) < 0;
        }
        public static bool CompareGrade(object a, object b)
        {
            Student s1 = (Student)a;
            Student s2 = (Student)b;
            return s1.grade > s2.grade;
        }
    }
}
```

Quick.cs:

```
using System;

namespace QuickSortProject
{
    class Quick
    {
        public delegate bool Comparison(object a, object b);
        public static void Sort(object[] array, int left, int right, Comparison
cmp)
        {
            if (left >= right) return;
            int last = Partition(array, left, right, cmp);
            Sort(array, left, last - 1, cmp);
            Sort(array, last + 1, right, cmp);
        }
        public static int Partition(object[] array, int left, int right,
Comparison cmp)
        {
            object pivot = array[left];
            int last = left;

            for (int i = left + 1; i <= right; i++)
            {
                if (cmp(array[i], pivot))
                {
                    last++;
                    Swap(array, i, last);
                }
            }
            Swap(array, left, last);
            return last;
        }
        static void Swap(object[] array, int first, int second)
        {
            object temp = array[first];
            array[first] = array[second];
            array[second] = temp;
        }
    }
}
```

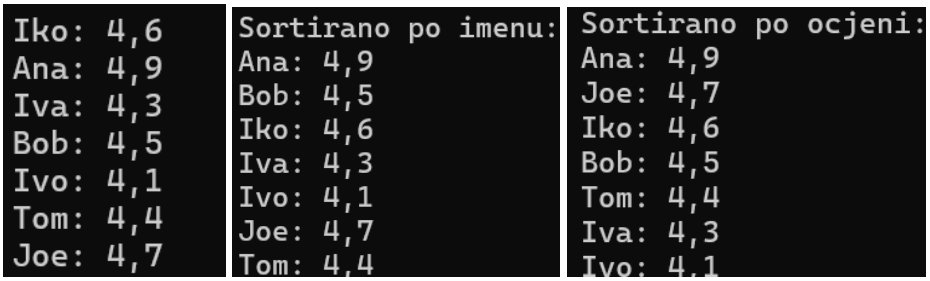
Program.cs:

```
using QuickSort;
using System;

namespace QuickSortProject
{
    class Program
    {
        static void Main(string[] args)
        {
            Student[] students = {
                new Student("Ivo", 4.1),
                new Student("Ana", 4.9),
                new Student("Iva", 4.3),
                new Student("Bob", 4.5),
                new Student("Joe", 4.7),
                new Student("Tom", 4.4),
                new Student("Iko", 4.6),
            };

            Console.WriteLine("Ispis studenata:");
            foreach (var s in students) Console.WriteLine(s);
            Console.WriteLine("\nParticionirano po imenu:");
            Quick.Partition(students, 0, students.Length - 1Student.CompareName);
            foreach (var s in students) Console.WriteLine(s);
            Console.WriteLine("\nSortirano po imenu:");
            Quick.Sort(students, 0, students.Length - 1, Student.CompareName);
            foreach (var s in students) Console.WriteLine(s);
            Console.WriteLine("\nSortirano po ocjeni:");
            Quick.Sort(students, 0, students.Length - 1, Student.CompareGrade);
            foreach (var s in students) Console.WriteLine(s);
        }
    }
}
```

Rezultat:



Glavni dio ovog zadatka je metoda *Partition*, koja odabire *pivot* (prvi element) i organizira niz tako da su elementi manji od *pivota* s lijeve strane, a veći s desne. Ukoliko je potrebno, elementi mijenjaju pozicije pomoću *Swap* metode. Delegat *Comparison* definiran je izvan klase i koristi se za prosljeđivanje kriterija. Omogućuje metodi *Sort* da koristi istu metodu za sortiranje studenata i po imenu i po ocjeni, ovisno o tome koja se metoda proslijedi. Unutar *Main-a* definira se niz studenata te se pozivom metode Partition prikazuje način na koji se studenti raspoređuju prema imenu. Zatim se primjenjuje metoda Sort kako bi se niz studenata u sortirao prema imenu i ocjeni.

Zadatak 3. – Priority Queues

U ovom zadatku implementira se prioritetni red koristeći strukturu heap. Najveći element uvijek se nalazi na vrhu heap-a i prvi se uklanja.

Heap.cs:

```
using System;
namespace PriorityQueue
{
    class Heap
    {
        private int size;
        private int last;
        private int[] priorities;
        public Heap(int size)
        {
            this.size = size;
            this.priorities = new int[size + 1];
            this.last = 0;
        }
        public void Insert(int i)
        {
            if (last == size)
                throw new Exception("pun");

            last++;
            priorities[last] = i;
            BubbleUp(last);
        }
        public int Retrieve()
        {
            if (last == 0)
                throw new Exception(" ");

            int max = priorities[1];
            priorities[1] = priorities[last];
            last--;
            BubbleDown(1);
            return max;
        }
        public void BubbleUp(int i)
        {
            if (i == 1) return;
            int parent = i / 2;
            if (priorities[i] > priorities[parent])
            {
                Swap(i, parent);
                BubbleUp(parent);
            }
        }
        public void BubbleDown(int i)
        {
            int left = 2 * i;
            int right = 2 * i + 1;
            int largest = i;

            if (left <= last && priorities[left] > priorities[largest])
                largest = left;

            if (right <= last && priorities[right] > priorities[largest])
                largest = right;

            if (largest != i)
            {
                Swap(i, largest);
                BubbleDown(largest);
            }
        }
        public void Swap(int i, int j)
```

Program.cs:

```
using PriorityQueue;
using System;
namespace PriorityQueue
{
    class Program
    {
        static void Main(string[] args)
        {
            Heap heap = new Heap(10);
            try
            {
                for (int i = 0; i < 11; i++)
                    heap.Insert(i);
            }
            catch (Exception x)
            {
                Console.WriteLine(x.Message);
            }
            heap.Display();
            try
            {
                while (true)
                {
                    int priority = heap.Retrieve();
                    heap.Display();
                }
            }
            catch (Exception x)
            {
                Console.WriteLine(x.Message);
            }
            Console.ReadKey();
        }
    }
}
```

Rezultat:

```
9 8 5 6 7 1 4 0 3 2
8 7 5 6 2 1 4 0 3
7 6 5 3 2 1 4 0
6 3 5 0 2 1 4
5 3 4 0 2 1
4 3 1 0 2
3 2 1 0
2 0 1
1 0
0
```

Heap je implementiran kao djelomično uređeno stablo gdje *roditelj* uvijek ima veću vrijednost od djece. U klasi *Heap* implementirana je struktura stabla u kojoj se provjerava položaj svakog elementa. Element se premješta u odnosu na svog *roditelja*. Cilj je održati pravilno uređen *heap*. Ukoliko je vrijednost elementa veća od *roditelja* oni zamjenjuju pozicije. Operacije koje koristimo su: *Insert*, *BubbleUp, Retrieve, BubbleDown*. *Insert* dodaje element na kraj i koristi metodu *BubbleUp* za njegovo pozicioniranje prema gore. *Retrieve* uklanja element s najvišim prioritetom tj. korijen stabla te ga mijenja sa posljednjim elementom. Zatim koristimo *BubbleDown* za vraćanje strukture *heap-a*. U *Main* programu se stvara objekt tipa *Heap*. Nakon unosa elemenata, pozivom metode *Display* ispisuje se trenutačno stanje *heap-a* na konzolu.

Zadatak 4. – Heap Sort

U ovom zadatku implementira se Heap Sort algoritam za sortiranje niza cijelih brojeva pomoću heap strukture podataka

Heap.cs:

```
using System;

namespace HeapSort
{
    class Heap
    {
        public static void Sort(int[] array)
        {
            Heapify(array);
            Arrange(array);
        }
        public static void Heapify(int[] array)
        {
            int n = array.Length;
            for (int i = n / 2 - 1; i >= 0; i--)
            {
                BubbleDown(array, i, n);
            }
        }
        public static void Arrange(int[] array)
        {
            int n = array.Length;
            for (int i = n - 1; i > 0; i--)
            {
                Swap(array, 0, i);
                BubbleDown(array, 0, i);
            }
        }
        public static void BubbleDown(int[] array, int i, int last)
        {
            int largest = i;
            int left = 2 * i + 1;
            int right = 2 * i + 2;
            if (left < last && array[left] > array[largest])
                largest = left;
            if (right < last && array[right] > array[largest])
                largest = right;
            if (largest != i)
            {
                Swap(array, i, largest);
                BubbleDown(array, largest, last);
            }
        }
        public static void Swap(int[] array, int i, int j)
        {
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }
}
```

Program.cs:

```
using System;

namespace HeapSort
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] array = { 5, 6, 8, 3, 4, 2, 1, 7, 3, 5, 4, 5, 4, 8 };
            foreach (var num in array) Console.Write(num + " ");
            Console.WriteLine();
            Heap.Sort(array);
            foreach (var num in array) Console.Write(num + " ");
            Console.WriteLine();
            Console.ReadKey();
        }
    }
}
```

Rezultat:

```
5 6 8 3 4 2 1 7 3 5 4 5 4 8
1 2 3 3 4 4 4 5 5 5 6 7 8 8
```

Klasa *Heap* u ovom slučaju, za razliku od prethodnog zadatka, sadrži statičke metode uključujući *Heapify*, koja pretvara nasumični niz u pravilno sortiran *heap* - preraspodjelom elemenata odozdo prema gore koristeći BubbleDown. U ovom zadatku nismo koristili metodu *BubbleUp*. Metoda *Arrange* sortira niz tako da uzastopno premješta najveći elemen *heap-a* na kraj niza i smanjuje veličinu samog *heap-a*, nakon svake zamjene imao ispravan poredak unutar *heap-a*. Metoda *Sort* potpuno sortira *heap* uzastopnim pozivanjem metoda *Heapify* i *Arrange* nad zadanim nizom. Unutar *Main-a* incijaliziramo niz te ga sortiramo pomoću već spomenutih metoda.