SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

Algoritmi i strukture podataka

Vježba 2.

Searching

Ivan Banovac, 220

Datum izvođenja vježbe:
10.11.2025

Zadatak 1. – Recursive Methods

U ovom zadatku implementiraju se i pozivaju rekurzivna i iterativna metoda, te se uspoređuje efikasnost i implementacija ove dvije metode

```
using System;

namespace Recursions
{
    class Program
    {
        static int Factorial(int n)
        {
            if (n <= 1) return 1;
            return n * Factorial(n - 1);
        }
        static int Fibonacci(int n)
        {
            if (n <= 0) return 0;
            if (n == 1) return 1;
            return Fibonacci(n - 1) + Fibonacci(n - 2);
        }
        static int FibonacciIter(int n)
        {
            if (n <= 0) return 0;
            if (n == 1) return 1;

            int a = 0;
            int b = 1;
            int result = 0;

            for (int i = 2; i <= n; i++)
            {
                result = a + b;
                a = b;
                b = result;
            }
            return result;
        }

        static void Main(string[] args)
        {
            int result = Factorial(6);
            Console.WriteLine("6!"+result);
            result = Fibonacci(6);
            Console.WriteLine("Fibonacci of 6="+result);
            result = FibonacciIter(6);
            Console.WriteLine("FibonacciIter of 6="+result);
        }
    }
}
```

Factorial(int n) računa faktorijel broja n metodom rekurzije. Prima argument n i ako je on <= 1 vraća vrijednost 1, u suprotnom se poziva rekurzivno na vrijednost n-1 te pomnoži sa n. Tim procesom dolazimo do faktorijela broja.

Fibonacci(int n) računa n-ti broj Fibonaccievog niza metodom rekurzije. Prima argument n i ako je <= 1 vraća 1, u suprotnom funkcija se poziva rekurzivno za n-1 i n-2 te se te vrijednosti zbrajaju. Tako dolazimo do Fibonaccijevog niza. fibonacciIter(int n) za razliku od prethodne koristi iterativnu metodu za izračun Fibonaccijevog niza.

Rekurzivnu metodu je vrlo jednostvano implementirati ali je neefikasan način za računanje. Iterativno do rezultata dolazimo brže jer se koristi obična petlja.

Rezultat:

```
6!720
Fibonacci of 6=8
FibonacciIter of 6=8
```

Zadatak 2. – Sequential and Binary search

U ovom zadatku implementiramo algoritme za pretraživanje, a to su Sequential Search i Binary Search.

```
using System;

namespace Searching
{
    class Program
    {
        static int SequentialSearch(int[] array, int value)
        {
            for (int i = 0; i < array.Length; i++)
            {
                if (array[i] == value)
                    return i;
            }
            return -1;
        }
        static int BinarySearch(int[] array, int value)
        {
            int low = 0;
            int high = array.Length - 1;

            while (low <= high)
            {
                int mid = (low + high) / 2;
                if (array[mid] == value)
                    return mid;
                else if (array[mid] < value)
                    low = mid + 1;
                else
                    high = mid - 1;
            }
            return -1;
        }
        static int BinarySearch(int[] array, int value, int low, int high)
        {
            if (low > high) return -1;

            int mid = (low + high) / 2;

            if (array[mid] == value)
                return mid;
            else if (array[mid] < value)
                return BinarySearch(array, value, mid + 1, high);
            else
                return BinarySearch(array, value, low, mid - 1);
        }

        static void Main(string[] args)
        {
            int[] array = { 1, 2, 3, 4, 5, 6, 7 };
            int index;
            Console.WriteLine("index:");
            index = SequentialSearch(array, 6);
            Console.WriteLine(index);
            Console.Write("index: ");
            index = BinarySearch(array, 6);
            Console.WriteLine(index);
            Console.Write("index: ");
            index = BinarySearch(array, 6, 0, array.Length - 1);
            Console.WriteLine(index);
        }
    }
}
```

Sequential Search koristi iteratitvnu metodu za pretraživanje elemenata u nesortiranom nizu – kompleksnost O(n) (*brute force*). Prima niz te prolazi kroz sve elemente dok ne dođe do traženog.

Binary Search koristi iterativnu i rekurzivnu metodu za pretraživanje elemenata u sortiranom nizu – kompleksnost O(log n). Ovakva metoda „dijeli“ niz na pola dok ne dođe do traženog elementa. Ukoliko se element ne nalazi na „sredini“, vraća high ili low vrijednost te se tako kreće u lijevo/desno kroz niz. Pamte se vrijednosti početnog/krajnjeg indeksa u podnizovima kako bi se postigla kompleksnost O(log n).

Rezultat:

```
index:
5
index: 5
index: 5
```

Zadatak 3. – Smart Arrays

U ovom zadatku implementirati će se niz Smart Array koji može promijeniti veličinu niza korištenjem indexera prilikom uklanjanja ili dodavanja elemenata u niz.

Program.cs:

```
using System;
using System.Collections;

namespace SmartArrays
{
    class Program
    {
        static void Main(string[] args)
        {
            SmartArray smarty = new SmartArray(6);

            for (int i = 0; i < 8; i++)
            {
                smarty.Add(i);
            }

            IEnumerator enumerator = smarty.GetEnumerator();
            while (enumerator.MoveNext())
            {
                int val = (int)enumerator.Current;
                Console.Write(val + " ");
            }
            Console.WriteLine();

            for (int i = 0; i < 8; i++)
            {
                IEnumerator e = smarty.GetEnumerator();
                while (e.MoveNext())
                {
                    Console.Write(e.Current + " ");
                }
                Console.WriteLine();

                Console.WriteLine("Removing " + i);
                smarty.Remove(i);
            }
        }
    }
}
```

SmartArray.cs:

```
using System;
using System.Collections;

namespace SmartArrays
{
    class SmartArray
    {
        int size;
        int last = -1;
        int[] array;
        public SmartArray(int initialSize)
        {
            this.size = initialSize;
            this.array = new int[size];
        }
        public int Length
        {
            get { return last + 1; }
        }
        public void Add(int item)
        {
            if (last == (size - 1))
            {
                int[] resized = new int[size * 2];
                Array.Copy(array, resized, size);
                array = resized;
                size = array.Length;
            }
            array[++last] = item;
        }
        public void Remove(int item)
        {
            int indexToRemove = -1;
            for (int i = 0; i <= last; i++)
            {
                if (array[i] == item)
                {
                    indexToRemove = i;
                    break;
                }
            }
            if (indexToRemove != -1)
            {
                for (int i = indexToRemove; i < last; i++)
                {
                    array[i] = array[i + 1];
                }
                last--;
            }
        }
        public int this[int index]
        {
            get { return array[index]; }
            set { array[index] = value; }
        }
        public IEnumerator GetEnumerator()
        {
            return new SmartEnumerator(this);
        }
        private class SmartEnumerator : IEnumerator, IDisposable
        {
            int index = -1;
            SmartArray smarty;
```

U klasi se nalaze - varijable size, last i array, te indexer this koji ima get i set metode za dohvaćanje i postavljanje vrijednosti niza na određenom indeksu.

Metoda Length dohvaća duljinu niza koja predstavlja indeks posljednjeg elementa uvećanog za jedan. Metoda Add i metoda Remove služe kako bi dodavali i uklanjali elemente iz niza. Unutar Add metode, kreiramo niz duplo veći od prethodnog ukoliko nema mjesta za dodavanje još jednog elementa.

Unutar Remove metode koristeći petlju tražimo željeni element, te ukoliko ga pronađemo niz se kopira u drugi niz metodom Copy.

Rezultat:

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