SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

Algoritmi i strukture podataka

Vježba 7.

Hash tables

Ivan Banovac, 220

Datum izvođenja vježbe:
22.12.2025

Zadatak 1. – Defining a Hash Table
Svi zadaci se nalaze u istom solution-u.

Node.cs:

```
using System;

namespace HashTable
{
    public class Node
    {
        public string Name { get; set; }
        public int Value { get; set; }
        public Node Next { get; set; }
        public Node(string name, int value, Node next)
        {
            Name = name;
            Value = value;
            Next = next;
        }
    }
}
```

HashTable.cs:

```
using System;

namespace HashTable
{
    public class HashTable
    {
        private Node[] buckets;
        private int length;
        public HashTable(int length)
        {
            this.length = length;
            buckets = new Node[length];
        }
        private int Hash(string str)
        {
            int hash = 0;
            foreach (char c in str)
            {
                hash += c;
            }
            return hash % length;
        }
        public void Insert(string name, int value)
        {
            int index = Hash(name);
            Node current = buckets[index];
            while (current != null)
            {
                if (current.Name == name)
                {
                    return;
                }
                current = current.Next;
            }
            buckets[index] = new Node(name, value, buckets[index]);
        }
        public int Search(string name)
        {
            int index = Hash(name);
            Node current = buckets[index];
            while (current != null)
            {
                if (current.Name == name)
                {
                    return current.Value;
                }
                current = current.Next;
            }
            throw new Exception("Node sa key-em nije pronađen u Hash-u");
        }
        public void Delete(string name)
        {
            int index = Hash(name);
            if (buckets[index] == null)
            {
                throw new Exception("Node sa key-em nije u Hash-u");
            }
            if (buckets[index].Name == name)
            {
                buckets[index] = buckets[index].Next;
                return;
            }
            Node current = buckets[index];
```

Program.cs:

```
using System;

namespace HashTable
{
    class Program
    {
        static void Main(string[] args)
        {
            HashTable table = new HashTable(9);

            table.Insert("Iva", 2);
            table.Insert("Kim", 3);
            table.Insert("Ana", 1);
            table.Insert("Tea", 9);
            table.Insert("Lea", 6);
            table.Insert("Ivo", 5);
            table.Insert("Bob", 4);
            table.Insert("Ena", 8);
            table.Insert("Joe", 7);
            table.Display();
            try
            {
                int val = table.Search("Bob");
                Console.WriteLine("Bob" + val);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            try
            {
                table.Delete("Bob");
                table.Delete("Joe");
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            table.Display();
        }
    }
}
```

Node.cs - Definira podatke, svaki čvor sadrži ime kao ključ, cjelobrojnu vrijednost i referencu na idući čvor, što omogućuje rješavanje kolizija metodom ulančavanja
HashTable.cs - Sadrži polje *buckets* koje je inicijalizirano na traženu veličinu u konstruktoru. Metoda *Hash* izračunava indeks unutar raspona [0,lenght-1].
Unutar Program.cs instanciramo tablicu veličine 9. Koriste se *try-catch* blokovei pri pretraživanju i brisanju kako bi sigurno obradila izminke u slučaju nepostojanja ključa.

Metodom *Insert* umećemo podatke unutar hash-a te ako dođe do kolizije (više imena ima isti hash), novi čvor se stavlja na početak liste na tom indeksu. Ako ime već postoji, operacija se zanemaruje.
Metoda *Search* služi za brzo pronalaženje vrijednosti pridruženoj odgovarajućem imenu (ključu). Metoda prvo poziva funkciju *Hash(name)* kako bi pretvorila *string(ime)* u cijeli broj koji odgovara indeksu u polju *buckets.* Program odlazi izravno na izračunatu poziciju u polju. Budući da hash tablica koristi ulančavanje, na tom se mjestu može nalaziti povezana lista

čvorova. Metoda zatim prolazi kroz povezanu listu čvor po čvor, uspoređujući traženo ime sa svojstvom *Name* svakog čvora. Ukoliko se pronađe čvor s istim imenom, metoda odmah vraća njegovu cjelobrojnu vrijednost *(Value)*. Ako metoda dođe do kraja liste bez podudaranja, baca se iznimka s porukom da element nije u tablici.
Metoda *Delete* uklanja čvorove unutar hash-a. Ukoliko je element prvi u listi, početak hash-a se stavlja na sljedeći element, a ukoliko se nalazi po sredini, pokazivač *Next* prethodnog elementa postavlja se na element koji slijedi nakon trenutnog elementa čime leminiramo traženi element (čvor).

Rezultati:

```
0
Iva2
1
Kim3
2
Ana1
3
Tea9
4
Lea6
5
Bob4Ivo
6
Ena8
7
Joe7
8

Bob4
0
Iva2
1
Kim3
2
Ana1
3
Tea9
4
Lea6
5
Ivo5
6
Ena8
7

8
```