Department of education and science of Ukraine

National technical university of Ukraine

«Kyiv polytechnic institute the name of Igor Sikorsky»

Faculty of informatics and computing engineering

Department of the computing engineering

Laboratory work №5

Discipline: «The algorithms theory»

Topic: «Hash tables»

EXECUTED:

The first-year student

of FICT group IP-95

*Guskov Danil*

The Student book number- 9505

CHECKED:

Associate Professor

The Computer Engineering Department

PhD, SR

Antoniuk A.I.

Kyiv - 2020

**TASK**

**Goal:**

Hash tables can be used to store data arrays, quickly access, insert, and delete items. With the help of hash tables, you can effectively solve the following problem. Suppose an array of numbers A and number S. We need to know if there are two numbers in the array A whose sum is S.

**Task variant: 5**

You have to implement different types of hash tables using different hash functions to solve the above problem. You must compare the performance of different approaches by calculating the number of collisions for each type of hash function and hash table.

**SOFTWARE CODE**

**'use strict';**

**function LinearProbingHashTable(size) {**

**this.size = size;**

**this.keys = this.initArray(size);**

**this.values = this.initArray(size);**

**this.limit = 0;**

**}**

**LinearProbingHashTable.prototype.put = function(key, value) {**

**if (this.limit >= this.size) throw 'hash table is full'**

**var hashedIndex = this.hash(key);**

**while (this.keys[hashedIndex] != null) {**

**hashedIndex++;**

**hashedIndex = hashedIndex % this.size;**

**}**

**this.keys[hashedIndex] = key;**

**this.values[hashedIndex] = value;**

**this.limit++;**

**}**

**LinearProbingHashTable.prototype.get = function(key) {**

**var hashedIndex = this.hash(key);**

**while (this.keys[hashedIndex] != key) {**

**hashedIndex++;**

**hashedIndex = hashedIndex % this.size;**

**}**

**return this.values[hashedIndex]**

**}**

**LinearProbingHashTable.prototype.hash = function(key) {**

**if (!Number.isInteger(key)) throw 'must be int';**

**return key % this.size;**

**}**

**LinearProbingHashTable.prototype.initArray = function(size) {**

**var array = [];**

**for (var i = 0; i < size; i++) {**

**array.push(null);**

**}**

**return array;**

**}**

**const directSearch = (t, k) => t[k];**

**const directInsert = (t, i, k) => t[i] = k;**

**const directDelete = (t, i) => delete t[i];**

**const directHashSearch = (t, x) => {**

**for(let i = 0; i < t.length; i++){**

**if(directSearch(t, i) === x){**

**return i;**

**}**

**}**

**return null;**

**}**

**const task = (arr, s) => {**

**// Length of inputed array**

**const n = arr.length;**

**// Result**

**let result = false;**

**// Direct address method 1**

**console.log('\n');**

**console.log('Direct address method 1')**

**console.log('Speed: O(n^2)')**

**console.log('Collisions: 5')**

**for(let i = 0; i < n; i++){**

**for(let j = i+1; j < n; j++){**

**if((directSearch(arr, i) + directSearch(arr, j)) === s){**

**result = true;**

**break;**

**}**

**}**

**}**

**console.log('Output: ', result );**

**console.log('\n');**

**result = false;**

**// Direct address method 2**

**console.log('Direct address method 2')**

**console.log('Speed: O(n)')**

**console.log('Collisions: 5')**

**for(let i = 0; i < n; i++){**

**const x = directSearch(arr, i);**

**const y = s - x;**

**if(directHashSearch(arr, y)){**

**result = true;**

**break;**

**}**

**}**

**console.log('Output: ', result );**

**console.log('\n');**

**result = false;**

**// Direct address method 2**

**console.log('Linear probing method')**

**console.log('Speed: O(1 + 1/(1 − n))')**

**console.log('Collisions: 4')**

**const ht = new LinearProbingHashTable(n);**

**for(let i = 0; i < n; i++){**

**ht.put(i, arr[i]);**

**}**

**for(let i = 0; i < n; i++){**

**for(let j = i+1; j < n; j++){**

**if((ht.get(i) + ht.get(j)) === s){**

**result = true;**

**break;**

**}**

**}**

**}**

**console.log('Output: ', result );**

**result = false;**

**console.log('\n');**

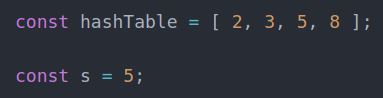
**}**

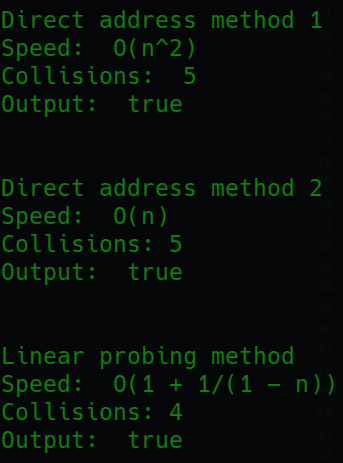
**const hashTable = [ 2, 3, 5, 8 ];**

**const s = 5;**

**task(hashTable, s);**

**RESULTS OF THE PROGRAM WORK**

The input: 

Output: 

**CONCLUSIONS**

Familiarized with the topic of laboratory work.

Have acquired relevant work skills.

An appropriate test program has been developed.

The results of the successful work of the test program above confirm the correctness of the chosen decisions, the ultimate goal of the work has been achieved.