Звіт

Автор: Момот Р. КІТ-119а

Дата: 03.05.2020

ЛАБОРАТОРНА РОБОТА № 10. АЛГОРИТМИ ПОШУКУ З ВИКОРИСТАННЯМ ТАБЛИЦЬ

Мета: закріпити знання про алгоритми пошуку, що вимагають додаткової пам’яті; одержати навички виконання операцій пошуку із використанням таблиць прямого доступу, справочників та хешованих таблиць.

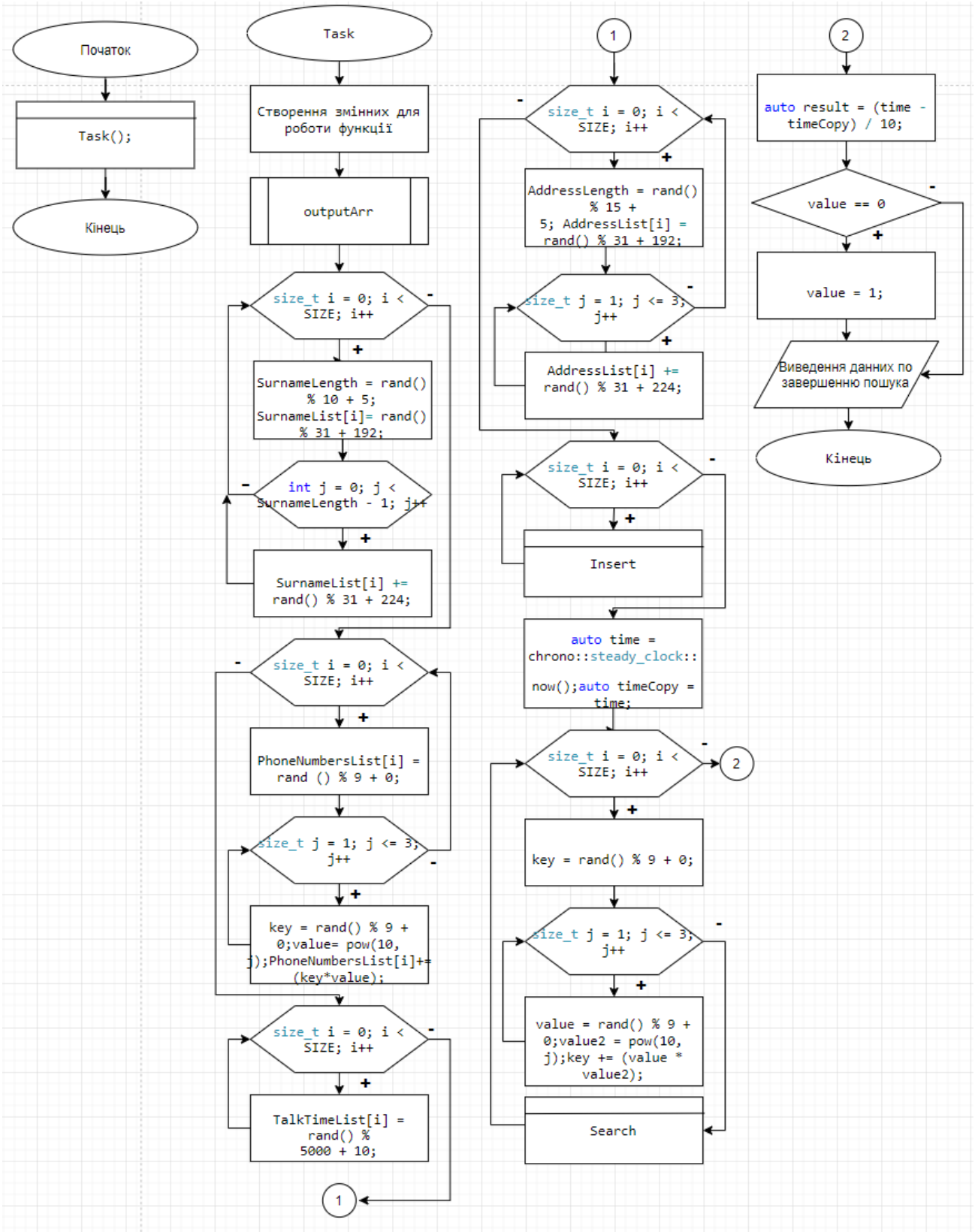
Індивідуальне завдання

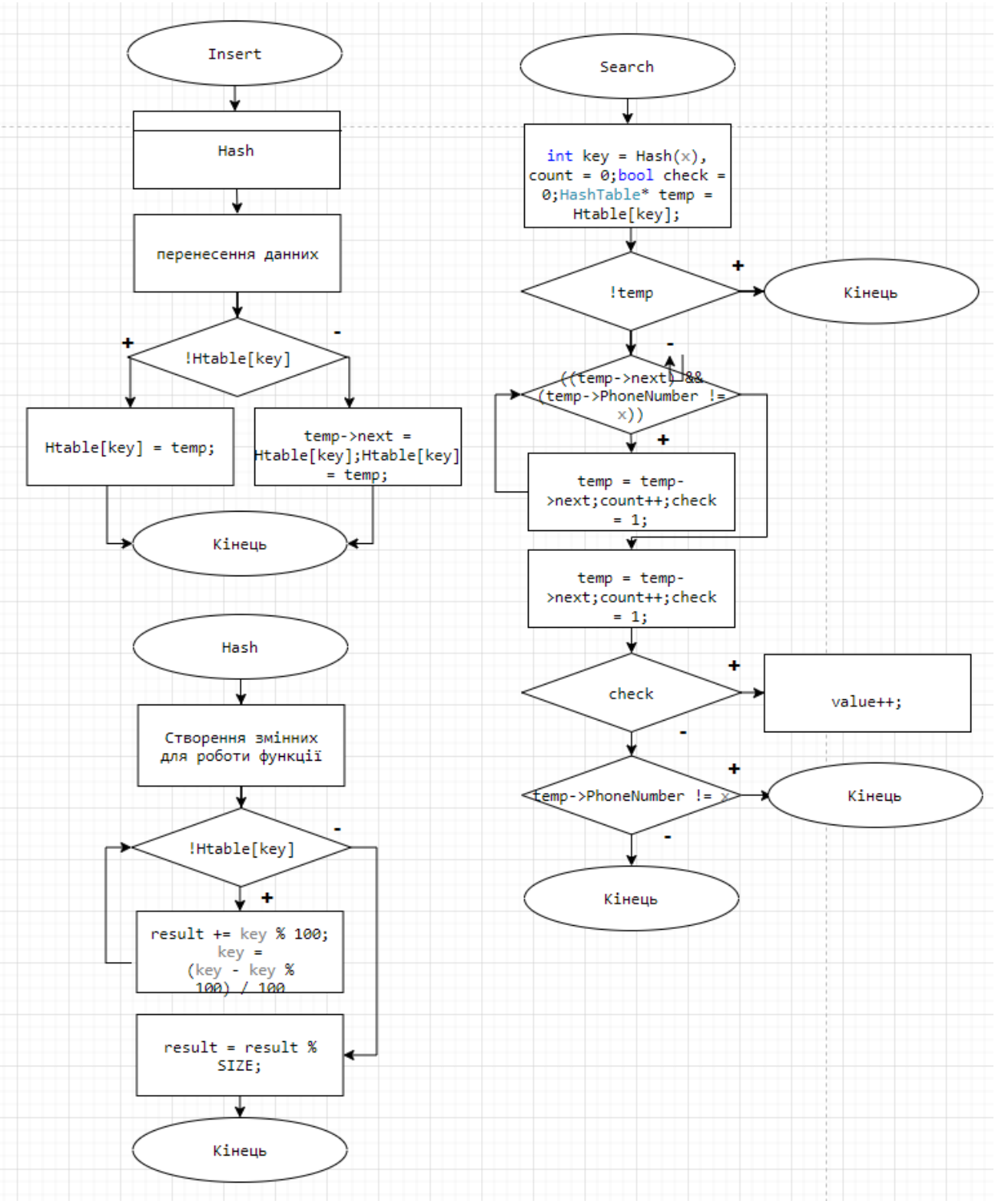
Для вмісту файла створити таблицю прямого доступу або хеш-таблицю у відповідності до індивідуального завдання. Перевірити працездатність створених таблиць на прикладі операцій пошуку.

Порівняти час пошуку із використанням створених таблиць та простих алгоритмів із лабораторної роботи №9. Для кожного з алгоритмів визначити кількість порівнянь у наборі даних з різною кількістю елементів (20, 1000, 5000, 10000, 50000) визначити час пошуку, заповнити таблицю по формі, побудувати графіки, зробити висновки.

N	Вміст вихідних даних	Таблиця	Хеш–функція	Ключ пошуку
13	Номер телефону, прізвище власника, адреса, час розмови	Хеш-таблиця з розподіленими ланцюжками переповнення	Функція середини квадрата	Номер телефону

Блок-схема алгоритму програми




Insert
Hash
перенесення данних
!Htable[key]
+
-
Htable[key] = temp;
temp->next = Htable[key];Htable[key] = temp;
Кінець
Hash
Створення змінних для роботи функції
!Htable[key]
-
+
result += key % 100; key = (key - key % 100) / 100
result = result % SIZE;
Кінець
Search
int key = Hash(x), count = 0;bool check = 0;HashTable* temp = Htable[key];
!temp
+
Кінець
-
((temp->next) && (temp->PhoneNumber != x))
+
temp = temp->next;count++;check = 1;
temp = temp->next;count++;check = 1;
check
+
value++;
-
temp->PhoneNumber != x
+
Кінець
-
Кінець

Текст програми

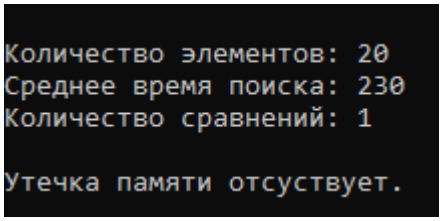
```
#include <iostream>
#include <iomanip>
#include <chrono>
#include <sstream>
#include <fstream>
#include <cmath>
using namespace std;
const int SIZE = 1000;                    //количество элементов в списке
static int amountOfComparisons = 0;
static int value = 0;

struct HashTable
{
	long long PhoneNumber;
	string Surname;
	int TalkTime;
	string Address;

	HashTable* next;
} *Htable[SIZE];

int Hash(int);
void TableInitialization();
HashTable* Search(int);
void Insert(long long, int, string, string);
void Task();
void DeleteTable();

void main()
{
	setlocale(LC_ALL, "Rus");
	srand(time(NULL));

	Task();

	if (_CrtDumpMemoryLeaks())
		cout << endl << "Есть утечка памяти." << endl;
	else
		cout << endl << "Утечка памяти отсуствует." << endl;

}

void Task()
{
	string* SurnameList = new string[SIZE];
	long long* PhoneNumbersList = new long long[SIZE];
	int* TalkTimeList = new int[SIZE];
	string* AddressList = new string[SIZE];
	long long key;
	int SurnameLength;
	int AddressLength;
	long long value, value2;

	TableInitialization();

	for (size_t i = 0; i < SIZE; i++)
	{
		SurnameLength = rand() % 10 + 5;

		SurnameList[i] = rand() % 31 + 192;
		for (int j = 0; j < SurnameLength - 1; j++)
			SurnameList[i] += rand() % 31 + 224;
	}
```

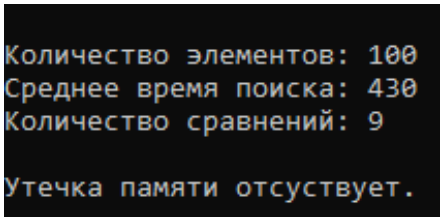
```
	for (size_t i = 0; i < SIZE; i++)
	{
		PhoneNumbersList[i] = rand () % 9 + 0;
		for (size_t j = 1; j <= 3; j++)
		{
			key = rand() % 9 + 0;
			value = pow(10, j);

			PhoneNumbersList[i] += (key*value);
		}
	}
	key = 0;

	for (size_t i = 0; i < SIZE; i++)
		TalkTimeList[i] = rand() % 5000 + 10;

	for (int i = 0; i < SIZE; i++)
	{
		AddressLength = rand() % 15 + 5;

		AddressList[i] = rand() % 31 + 192;
		for (int j = 0; j < AddressLength - 1; j++)
			AddressList[i] += rand() % 31 + 224;
	}

	for (size_t i = 0; i < SIZE; i++)
		Insert(PhoneNumbersList[i], TalkTimeList[i], SurnameList[i], AddressList[i]);

	auto time = chrono::steady_clock::now();
	auto timeCopy = time;
	for (size_t i = 0; i < 10; i++)
	{
		key = rand() % 9 + 0;
		for (size_t j = 1; j <= 3; j++)
		{
			value = rand() % 9 + 0;
			value2 = pow(10, j);

			key += (value * value2);
		}

		auto begin = chrono::steady_clock::now();
		Search(key);
		auto end = chrono::steady_clock::now();
		auto elapsed_ms = chrono::duration_cast<chrono::nanoseconds>(end - begin);
		time += elapsed_ms;
	}
	auto result = (time - timeCopy) / 10;
	if (value == 0) value = 1;

	cout << "\nКоличество элементов: " << SIZE << "\nСреднее время поиска: " <<
result.count() << endl;
	cout << "Количество сравнений: " << amountOfComparisons / value << endl;

	delete[] SurnameList;
	delete[] PhoneNumbersList;
	delete[] TalkTimeList;
	delete[] AddressList;
	DeleteTable();
}

void TableInitialization()
{
	for (int i = 0; i < SIZE; i++)
		Htable[i] = NULL;
```

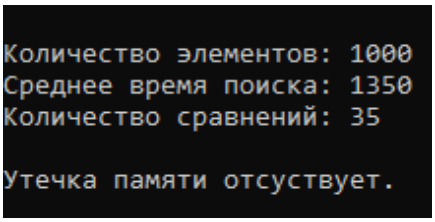
```
}

void Insert(long long phone, int talkTime, string surname, string address)
{
	HashTable* temp = new HashTable;
	long long key = Hash(phone);

	temp->next = NULL;
	temp->PhoneNumber = phone;
	temp->TalkTime = talkTime;
	temp->Surname = surname;
	temp->Address = address;

	//cout << setw(3) << key << "  ";
	//cout << temp->PhoneNumber << "\t" << temp->TalkTime << "\t" << temp->Surname << "\t"
<< temp->Address << endl;

	if (!Htable[key]) Htable[key] = temp;
	else
	{
		temp->next = Htable[key];
		Htable[key] = temp;
	}
}

int Hash(int key)
{
	key *= key;		// возвести ключ в квадрат
	int result = 0;
	while (key > 0)
	{
		result += key % 100;
		key = (key - key % 100) / 100;
	}
	result = result % SIZE;
	return result;

}

HashTable* Search(int x)
{
	int key = Hash(x), count = 0;
	bool check = 0;
	HashTable* temp = Htable[key];

	if (!temp) return NULL;
	while ((temp->next) && (temp->PhoneNumber != x))
	{
		temp = temp->next;
		count++;
		check = 1;
	}
	amountOfComparisons += count + 1;
	if (check) value++;
	if (temp->PhoneNumber != x) return NULL;

	return temp;
}

void DeleteTable()
{
	struct HashTable* p, * pCopy;

	for (size_t i = 0; i < SIZE; i++)
	{
		p = Htable[i];
```

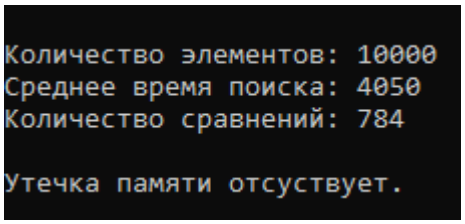
```
            if (Htable[i] != NULL)
                while (p)
                {
                    pCopy = p;
                    p = p->next;
                    delete pCopy;
                }
        }
}
```

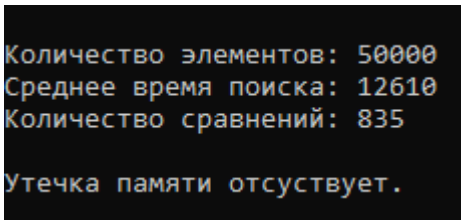
Результати роботи програми











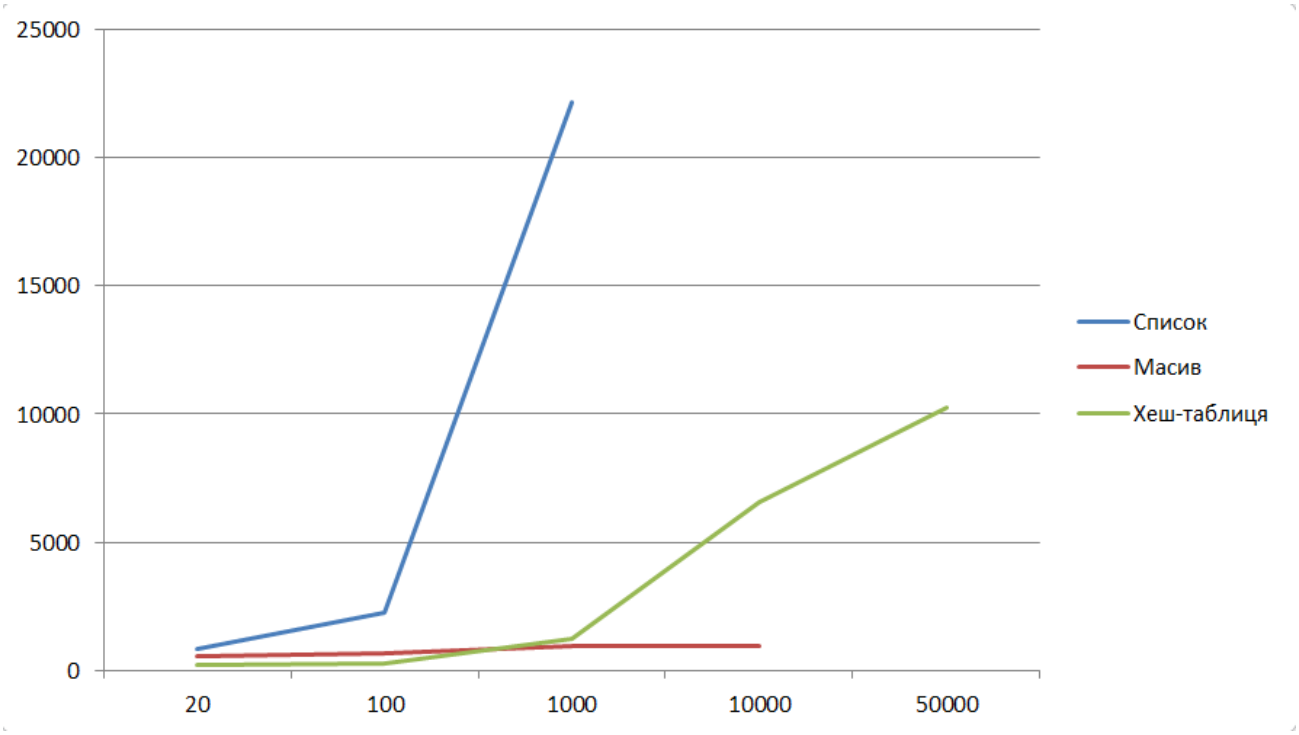
Результати тестування алгоритма пошуку хеш-таблиці

Кількість елементів	20	100	1000	10000	50000
Кількість порівнянь	1	7	30	1141	825
Час пошуку*	230	250	1220	6590	10220

*Час пошуку вказан у наносекундах.

**У таблиці вказані середні значення після багаторазових перевірок даних.

Графік часу роботи алгоритмів пошуку



Висновок

У результаті роботи програми було розроблено хеш–таблицю із розподіленими ланцюжками. Хеш–функцією була функція середини квадрата, ключем пошуку – номер телефону. Було допущено спрощення, що телефон містить лише 4 цифри, так як при зведенні чисел, які містять 10 цифр, розміру змінних, передбачених C++ не вистачало. Як можна бачити із графіку вище хеш-таблиця працює швидше за списка і масива. Це відбувається тому що, у таблиці хеш–функція рівномірно розподіляє елементи, а у випадку виникнення колізій створюється лінійний список за вказаною адресою (індексом). При пошуку елементу таблиця, на відміну від лінійного пошуку з бар'єром, знає за якою адресою потрібно шукати елемент, а не перевіряє інші елементи таблиці, які не треба шукати. І навіть, якщо шукомий елемент знаходиться поза таблицею, то той однозв'язний список, у якому відбувається пошук у таблиці, все одно менший за просто однозв'язний список або масив із лінійним пошуком.

Відповіді на питання

1. Що записується в хеш–таблицю?

В хеш–таблицю записуються ключ та адреса запису.

2. Чим визначається індекс запису в хеш-таблиці?

Індекс визначається за формулою $r = H(key)$, де r - індекс, key – ключ пошуку, а H – деяка функція, яка перетворює ключ пошуку в індекс (адресу запису).

3. Які основні проблеми хешування і в чому вони полягають?

Колізії (або переповнення) – основна проблема хешованих таблиць. Вона полягає в тому, що різні ключі можуть відображатися в одну і ту ж адресу.

4. Скільки операцій порівняння виконується при пошуку по ключу із застосуванням таблиць прямого доступу?

При пошуку по ключу із застосуванням таблиць прямого доступу виконується лише одна операція порівняння.

5. Які ключі називають синонімами?

Синонімами називають різні ключі, які відображаються в одну і ту ж адресу.

6. Які недоліки алгоритму пошуку по ключу з використанням таблиць прямого доступу?

Розміри таблиць прямого доступу можуть бути дуже великими, при тому, що більша частина такої таблиці буде пуста бо фактична множина не буде покривати всі можливі варіанти значень ключів.

7. Яке призначення хеш-функції?

Хеш-функція перетворює ключ у адресу, за якою буде зберігатися цей ключ.

8. Назовіть базові функції хешування.

Ділення по модулю, середина квадрату, згортка, перетворення системи числення.

9. Як створюється хеш-таблиця при реалізації метода відкритої адресації? Що зберігається в елементах такої хеш-таблиці?

При використанні методу відкритої адресації всі елементи зберігаються безпосередньо в хеш-таблиці, тобто кожен запис таблиці містить або елемент динамічної множини, або значення NULL.

При виконанні запису в хеш-таблицю звернення виконується до слоту, індекс якого визначено хеш-кодом ключа пошуку. Якщо при спробі запису в таблицю виявляється, що необхідне місце в таблиці вже зайнято, то значення записується в ту ж таблицю але на якесь інше місце. Інше місце визначається за допомогою вторинної функції хешування H', аргументом якої в загальному випадку може бути і вихідне значення ключа і результат попереднього хешування: ***r = H '(key, r`)***, де ***r`*** - адреса, отримана при попередньому застосуванні функції хешування. Якщо отримана в результаті застосування функції H' адреса також виявляється зайнятою, то функція H' застосовується повторно - до тих пір, поки не буде знайдено вільне місце.

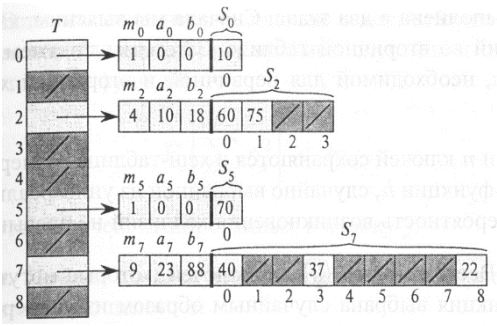
10. Як створюється таблиця прямого доступу? Що зберігається в її елементах?

При створенні таблиці прямого доступу виділяється пам'ять для зберігання всієї таблиці і заповнюється порожніми записами. Потім записи вносяться в таблицю - кожна на своє місце, яке визначається ключем цього запису.

11. Назовіть алгоритми розв’язку колізій при відкритій адресації.

- Лінійне дослідження
- Квадратичне дослідження
- Подвійне дослідження
- Пакетування (використовується як доповнення до інших досліджень)

12. Яке хешування зображено на даному рисунку?



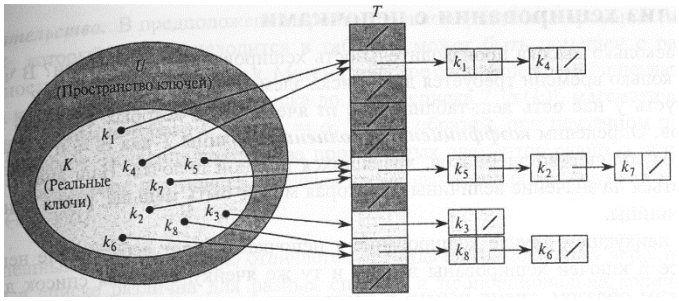
На рисунку зображена схема ідеального хешування.

13. В чому суть метода розв’язку колізій при використанні розподілених ланцюгів переповнень?

У структуру кожного запису додається ще одне поле - покажчик на наступний запис. Через ці покажчики записи з ключами-синонімами зв'язуються в лінійний список, початок якого знаходиться в основній таблиці, а продовження - поза нею. При вставці запису в таблицю по функції хешування обчислюється адреса записи (або пакета) в основній таблиці.

Якщо це місце в основній таблиці вільно, то запис заноситься в основну таблицю. Якщо ж місце в основній таблиці зайнято, то запис розташовується поза нею.

14. Який метод хешування зображений на рисунку?



На рисунку зображен метод хешування з використанням розподілених ланцюгів переповнень

15. Чому пошук із використанням таблиць прямого доступу широко не впроваджується ?

Таблиці прямого доступу можуть застосовуватися тільки і для таких завдань, в яких розмір простору записів може бути дорівнює розміру простору ключів. У більшості реальних задач, однак, розмір простору записів багато менше, ніж простору ключів.