Завдання до комп’ютерного практикуму № 2
з дисципліни «Алгоритми та структури даних»
Тема: «Хешування»

Мета роботи

Отримати навички роботи з алгоритмами організації хеш-таблиць з прямою адресацією та хеш-таблиць з вирішенням конфліктів (колізій) методом ланцюгів.

Основні теоретичні відомості

Хешування (HASHING) – це пошук, який принципово відрізняється від методів пошуку, що мають в основі порівняння значень ключів елементів. Замість переміщення по структурі даних з порівнянням ключа пошуку з ключем в елементі, відбувається спроба звернення до потрібного елемента (що зберігається в хеш-таблиці) за рахунок виконання арифметичних операцій над його ключем.

Хеш-таблиця – це зазвичай масив, розмір якого пропорційний реальній кількості елементів, що зберігаються.

Для обчислення адреси елемента в хеш-таблиці використовується хеш-функція h. Функція h відображає сукупність ключів у адреси хеш-таблиці.

1. Алгоритми пошуку (хешування)

I крок. Обчислення хеш-функції, що перетворює ключ пошуку в адресу в хеш-таблиці.

В ідеальному випадку різні ключі мають відображатись у різні адреси, однак, часто два та більше різних ключів можуть перетворюються в одну і ту саму адресу – звідси II крок.

II крок. Вирішення конфліктів – процес, що обробляє ключі, які перетворюються в одну і ту саму адресу хеш-таблиці.

2. Хеш-таблиці з прямою адресацією

Хеш-функція: *h(key) = key* (ключ елемента використовують як адресу).

Переваги: миттєве звернення до елементів, O(1).

Недоліки: не всі ключі можна інтерпретувати як індекси (від'ємні, з плаваючою точкою, слова тощо); розмір таблиці залежить від діапазону значень ключів елементів, що необхідно обробити; кількість елементів, що зберігається, може бути набагато меншою за сукупність ключів; значення в послідовності мають бути різними.

Приклад структури елемента:

```
#define KEY int
typedef struct {
    KEY * key;
} item;
```

Приклад структури хеш-таблиці:

```
typedef struct {
    int n;
    item ** array;
} hash_table;
```

Блок-схема алгоритму ініціалізації хеш-таблиці (функція DA_INIT(T, x)[1]), рис. 1, де:

T – вказівник на хеш-таблицю (масив);

MAX – максимальне значення ключа елемента + 1, розмір хеш-таблиці залежить від діапазону значень ключів елементів.

[1] DA – DIRECT ADDRESS

2.1. Операції

2.1.1. Додавання елемента в хеш-таблицю, функція DA_INSERT(T, x)

У комірку масиву *T* з індексом *x->key* (ключ елемента) записати вказівник на елемент, рис. 2 а), де:

T – вказівник на хеш-таблицю (масив);

x – вказівник на елемент, що необхідно додати.

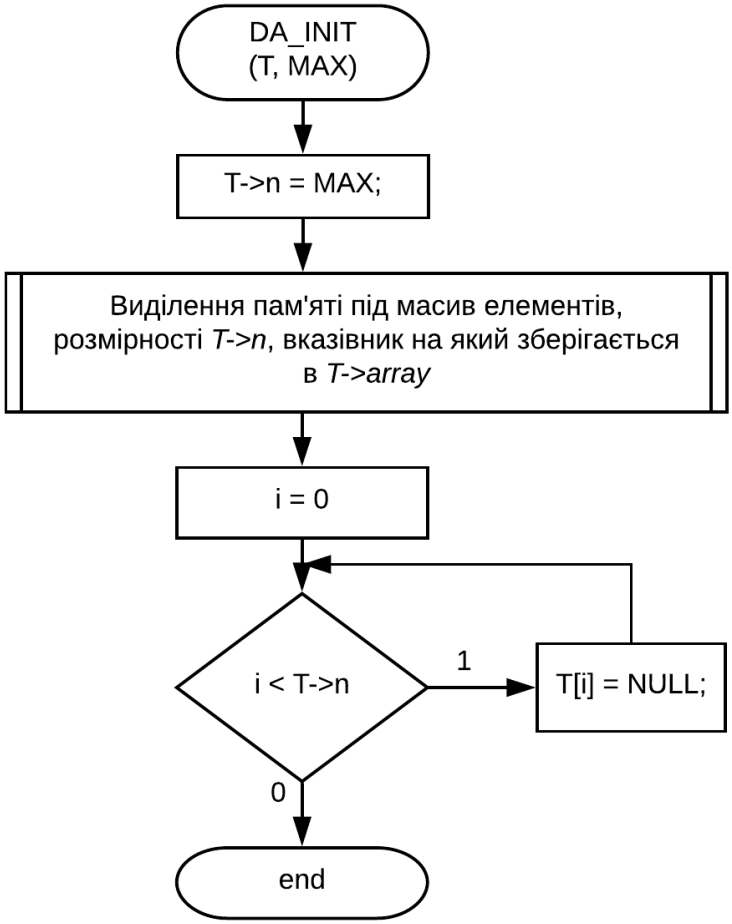

Рис. 1. Блок-схема ініціалізації хеш-таблиці

2.1.2. Видалення елемента з хеш-таблиці, функція DA_DELETE(T, x)

У комірку масиву *T* з індексом *x->key* (ключ елемента) записати нульовий вказівник, рис. 2 б), де:

T – вказівник на хеш-таблицю (масив);

x – вказівник на елемент, що необхідно видалити.

Увага! Можливий витік пам’яті – вивільнення пам’яті, що була виділена під елемент, необхідно передбачити окремо.

2.1.3. Пошук елемента за ключем, функція DA_SEARCH(T, key)

Повернути вказівник на елемент, що зберігається в комірці з індексом *key* масиву *T*, рис. 2 в), де:

T – вказівник на хеш-таблицю (масив);

key – ключ-пошуку елемента.

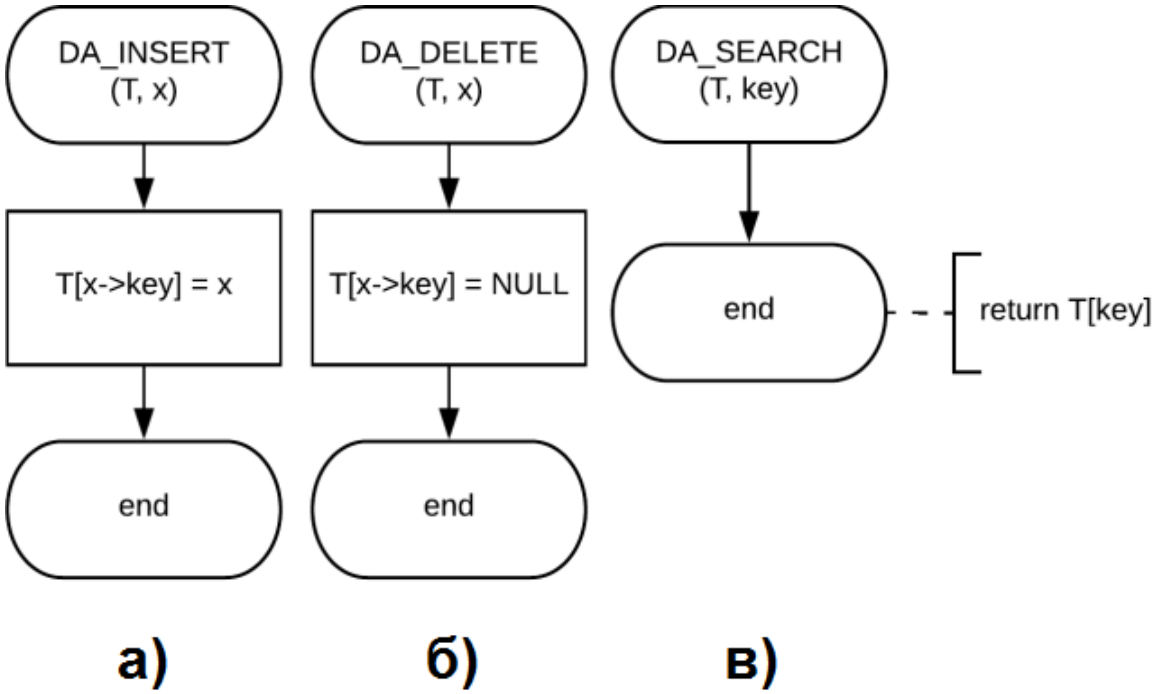

Рис. 2. Блок-схеми алгоритмів організації хеш-таблиць з прямою адресацією

3. Метод ланцюгів (роздільне зв’язування)

При вирішенні конфліктів (collision) методом ланцюгів, елементи, що були хешовані в одну адресу таблиці, зв’язуємо у список, рис. 3.

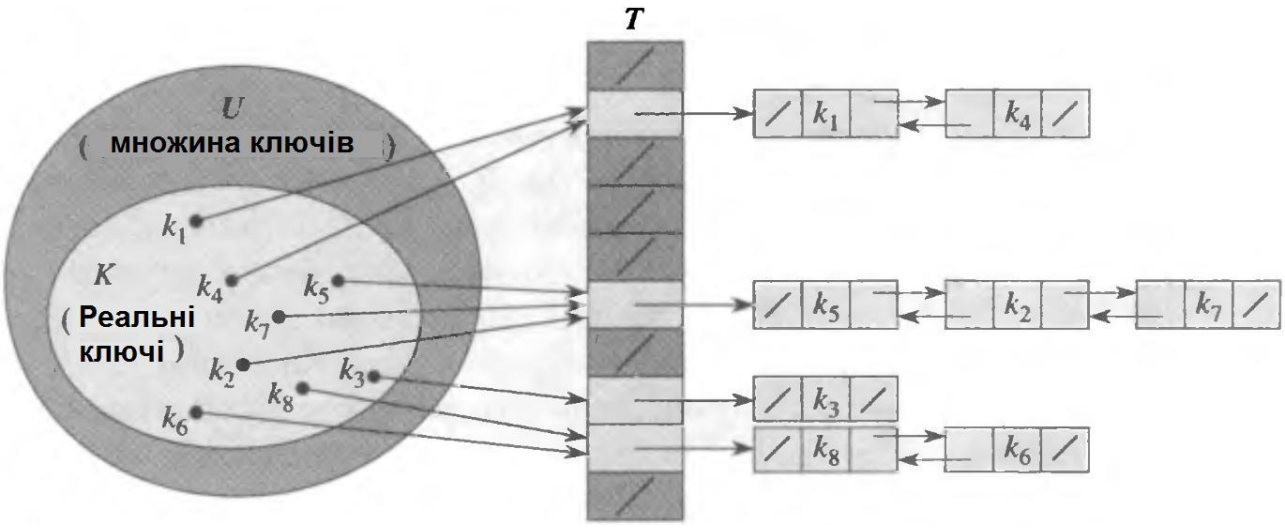

Рис. 3. Метод ланцюгів, вирішення конфліктів.

Перевага: економія пам'яті.

Недоліки: пошук у списку пропорційний довжині списку; використання вказівників.

3.1. Операції

3.1.1. Додавання елемента в хеш-таблицю, функція CH_INSERT(T, x)[2]

Додати елемент *x* у голову списку, вказівник на який зберігається у комірці масиву *T* з індексом *h(x->key)*, рис. 4 а), де:

T – вказівник на хеш-таблицю (масив);

x – вказівник на елемент, що необхідно додати.

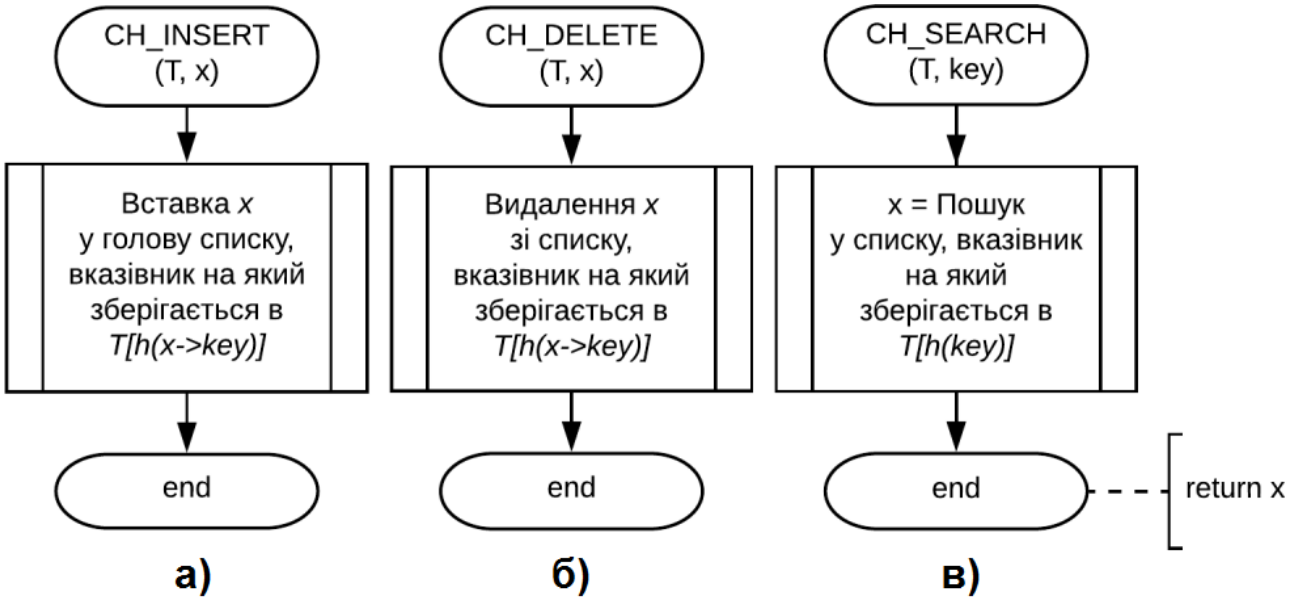

Рис. 4. Блок-схеми алгоритмів організації хеш-таблиць з вирішенням конфліктів (колізій) методом ланцюгів

3.1.2. Видалення елемента з хеш-таблиці, функція CH_DELETE(T, x)

Видалити елемент *x* зі списку, вказівник на голову якого зберігається у комірці масиву *T* з індексом *h(x->key)*, рис. 4 б), де:

T – вказівник на хеш-таблицю (масив);

x – вказівник на елемент, що необхідно видалити.

Увага! Можливий витік пам’яті – вивільнення пам’яті, що була виділена під елемент, необхідно передбачити окремо.

[2] CH – CHAINED HASH

3.1.3. Пошук елемента за ключем, функція CH_SEARCH(T, key)

Пошук елемента з ключем *key* у списку, вказівник на голову якого зберігається у комірці масиву *T* з індексом *h(x->key)*, рис. 4 в), де:

T – вказівник на хеш-таблицю (масив);

key – ключ-пошуку елемента.

Строк виконання комп’ютерного практикуму № 2

10 квітня 2019 року

Порядок виконання роботи

1. Написати програму, відповідно варіанту завдання, табл. 3.
2. Згенерувати послідовність із N елементів, у хеш-таблицю, отримавши мінімальну, середню та максимальну кількість порівнянь і копіювань.

   Для організації хеш-таблиці з прямою адресацією передбачити перевірку, чи елемент вже зберігається, із відповідним повідомленням.

   Для організації хеш-таблиці методом ланцюгів додатково згенерувати послідовність із 100 елементів, у нову хеш-таблицю (розмірності 100), отримавши мінімальну, середню та максимальну кількість порівнянь і копіювань.
3. Реалізувати пошук N випадкових елементів із заданого діапазону, отримавши мінімальну, середню та максимальну кількість порівнянь і копіювань.
4. Реалізувати видалення N випадкових елементів із заданого діапазону, отримавши мінімальну, середню та максимальну кількість порівнянь і копіювань.
5. Результати оформити в звіт:
   1. Титульний лист.
   2. Варіант завдання.
   3. Завдання.
   4. Лістинг програми.
   6. Результати роботи програми у вигляді таблиць (табл. 1, 2)
   7. Висновки.

Табл. 1. Результати роботи програми, кількість порівнянь

Організація хеш-таблиці	N	Розмір хеш-таблиці, байт	Додатково використана пам'ять, байт	Порівнянь								
				Додавання			Видалення			Пошук		
				мін	сер	макс	мін	сер	макс	мін	сер	макс

Табл. 2. Результати роботи програми, кількість копіювань

Організація хеш-таблиці	N	Розмір хеш-таблиці, байт	Додатково використана пам'ять,байт	Копіювань								
				Додавання			Видалення			Пошук		
				мін	сер	макс	мін	сер	макс	мін	сер	макс

Табл. 3. Варіанти завдань

№	Організація хеш-таблиці	Кількість елементів, *N*	Хеш-функція, *h*	Діапазон значень
1	З прямою адресацією	401	*h(key) = key*	0..20000
	Метод ланцюгів		*h(key) = key % N*	
2	З прямою адресацією	101	*h(key) = key*	0..11000
	Метод ланцюгів		*h(key) = key % N*	
3	З прямою адресацією	137	*h(key) = key*	0..25000
	Метод ланцюгів		*h(key) = key % N*	
4	З прямою адресацією	107	*h(key) = key*	0..32000
	Метод ланцюгів		*h(key) = key % N*	
5	З прямою адресацією	821	*h(key) = key*	0..18000
	Метод ланцюгів		*h(key) = key % N*	
6	З прямою адресацією	277	*h(key) = key*	0..29000
	Метод ланцюгів		*h(key) = key % N*	
7	З прямою адресацією	89	*h(key) = key*	0..8000
	Метод ланцюгів		*h(key) = key % N*	
8	З прямою адресацією	223	*h(key) = key*	0..3000
	Метод ланцюгів		*h(key) = key % N*	
9	З прямою адресацією	373	*h(key) = key*	0..15000
	Метод ланцюгів		*h(key) = key % N*	
10	З прямою адресацією	677	*h(key) = key*	0..26000
	Метод ланцюгів		*h(key) = key % N*	
11	З прямою адресацією	523	*h(key) = key*	0..30000
	Метод ланцюгів		*h(key) = key % N*	
12	З прямою адресацією	599	*h(key) = key*	0..9000
	Метод ланцюгів		*h(key) = key % N*	
13	З прямою адресацією	173	*h(key) = key*	0..2000
	Метод ланцюгів		*h(key) = key % N*	

Продовж. табл. 3.

№	Організація хеш-таблиці	Кількість елементів, *N*	Хеш-функція, *h*	Діапазон значень
14	З прямою адресацією	263	*h(key) = key*	0..24000
	Метод ланцюгів		*h(key) = key % N*	
15	З прямою адресацією	307	*h(key) = key*	0..16000
	Метод ланцюгів		*h(key) = key % N*	
16	З прямою адресацією	727	*h(key) = key*	0..5000
	Метод ланцюгів		*h(key) = key % N*	
17	З прямою адресацією	853	*h(key) = key*	0..7000
	Метод ланцюгів		*h(key) = key % N*	
18	З прямою адресацією	443	*h(key) = key*	0..10000
	Метод ланцюгів		*h(key) = key % N*	
19	З прямою адресацією	181	*h(key) = key*	0..27000
	Метод ланцюгів		*h(key) = key % N*	
20	З прямою адресацією	97	*h(key) = key*	0..1000
	Метод ланцюгів		*h(key) = key % N*	
21	З прямою адресацією	227	*h(key) = key*	0..31000
	Метод ланцюгів		*h(key) = key % N*	
22	З прямою адресацією	523	*h(key) = key*	0..13000
	Метод ланцюгів		*h(key) = key % N*	
23	З прямою адресацією	823	*h(key) = key*	0..6000
	Метод ланцюгів		*h(key) = key % N*	
24	З прямою адресацією	337	*h(key) = key*	0..17000
	Метод ланцюгів		*h(key) = key % N*	
25	З прямою адресацією	499	*h(key) = key*	0..21000
	Метод ланцюгів		*h(key) = key % N*	
26	З прямою адресацією	557	*h(key) = key*	0..28000
	Метод ланцюгів		*h(key) = key % N*	
27	З прямою адресацією	601	*h(key) = key*	0..14000
	Метод ланцюгів		*h(key) = key % N*	
28	З прямою адресацією	389	*h(key) = key*	0..23000
	Метод ланцюгів		*h(key) = key % N*	
29	З прямою адресацією	463	*h(key) = key*	0..4000
	Метод ланцюгів		*h(key) = key % N*	
30	З прямою адресацією	907	*h(key) = key*	0..19000
	Метод ланцюгів		*h(key) = key % N*	
31	З прямою адресацією	293	*h(key) = key*	0..12000
	Метод ланцюгів		*h(key) = key % N*	
32	З прямою адресацією	193	*h(key) = key*	0..22000
	Метод ланцюгів		*h(key) = key % N*	