

Література

1. **Wasserverbrauch** und Abwassereinigung. Umweltbericht für den Kanton Zürich 2004. – 4 p.
2. **Кульський Л.А., Строкач П.П.** Технология очистки природных вод. – К.: Высш. шк. – 1986. – 48 с.
3. **Nasseri S. Vaezi F., Mahvi A.H., Nabizadeh R., Haddadi S.** Determination of the ultrasonic effectiveness in advanced wastewater treatment. J. Environ. Health. Sci. Eng. 2006, vol. 3, № 2, pp. 109-116.
4. **Tsukamoto I. Yim B., Stavarache C.E., Furuta M., Hashiba K., Maeda Y.** Inactivation of *Saccharomyces cerevisiae* by ultrasonic irradiation// Ultrasonics Sonochemistry, vol. 11 (2004) pp. 61-65.
5. **Preeti C. Sangave, Aniruddha B. Pandit.** Ultrasound and enzyme assisted biodegradation of distillery wastewater. J. of Environmental Management. Vol. 80, Issue 1, 2006, pp. 36-46.

УДК 681.3.06 *Студ. А.В. Нечепуренко; доц. Ю.І. Грицюк, канд. техн. наук – НЛТУ України, м. Львів*

ЗАСТОСУВАННЯ ЗМІШАНИХ ТАБЛИЦЬ ДЛЯ РОБОТИ З ЕЛЕМЕНТАМИ БУЛЕВИХ МАТРИЦЬ

Розглянуто питання застосування динамічної структури даних – *змішаних таблиць* (hash tables) для роботи з великими масивами, значна більшість елементів яких не використовується або має фіксовані значення. Здатність змішаних таблиць оптимізувати доступ до таких даних робить їх надзвичайно корисними в багатьох ситуаціях, які виникають під час розроблення сучасних програмних продуктів. До них належать задачі, пов'язані з аналізом розріджених масивів, які представляють, наприклад, булеві матриці суміжності чи інцидентності. Завдяки використанню змішаних таблиць пам'ять для зберігання кожного з бінарних елементів виділяється з пули вільної пам'яті комп'ютера тільки в міру потреби.

Stud. A.V. Nechepurenko, assist. prof. Yu.I. Grytcyuk – NUFWT of Ukraine, L'viv

Application of the mixed tables is for storage of elements of boole matrices

The question of application of run-time structure of information is considered – *the mixed tables* (hash tables) for work with large arrays, considerable most elements of which are not utilized or have the fixed values. To optimize ability of the mixed tables access to such information does them extraordinarily useful in many situations which arise up during development of modern software products. Tasks, related to the analysis of disperse an array, which present, for example, the boole matrices of contiguity or incident, belong to them. Due to the use of the mixed tables memory is for storage each of binary elements selected from pools of free space of computer only to the extent of necessity.

1. Постановка завдання

Для роботи елементами булевих матриць, особливо якщо ці матриці мають великі розміри, існує проблема компактного їх зберігання, а також ефективного пошуку потрібних з них. Ефективність програмної реалізації цієї задачі залежить від організації структури даних і вибраної методики їх оброблення. Основою такої програми є інформаційно-пошукова система, призначеної для створення бази даних, її редагування та оброблення. Пошук потрібного запису у базі здійснюється шляхом порівняння заданого ключового значення з ключами кожного елемента структури даних. Коли знайдено збіжний ключ, то запис можна зчитувати, потім його редагувати чи обробляти. Широко відома на сьогодні динамічна структура як бінарні дерева [2, ст. 122] хоча і дає змогу ефективно здійснювати будь-який пошук, проте ці дерева

придатні тільки для роботи з відсортованими даними за рахунок значного зменшення необхідної кількості порівнянь ключів.

На противагу бінарним деревам записи, які зберігаються у звичайному масиві (чий індекс є аналогом ключа), можуть бути знайдені дуже швидко і без проведення будь-яких порівнянь. Якщо відомі ключ (тобто індекс масиву), розмір запису і початкова адреса масиву, то обчислення адреси потрібного запису відбувається за допомогою множення і додавання. Незважаючи на те, що звичайні масиви (одно-, дво- чи навіть багатовимірні) дають нам такі переваги, проте часто трапляються задачі (наприклад, булеві матриці великих розмірів), коли необхідно зберігати декілька сотень записів з діапазоном індексів від одного до декількох тисяч і тільки декілька сотень елементів є значущими. У цьому випадку матриця, яка складається з декількох мільйонів елементів, на 99,99 % є порожньою, тобто будь-яка традиційна пошукова система для знаходження тих чи інших записів вимагатиме значних затрат часу. Динамічні структури даних, серед яких змішані таблиці зокрема, якраз і призначені для вирішення таких неоднозначних ситуацій шляхом ефективного обчислення відповідних індексів масиву – одно-, дво- чи багатовимірного.

2. Особливості проектування змішаних таблиць

Змішані таблиці переважно реалізують у вигляді одновимірного масиву фіксованого розміру [3, ст. 104], елементами якого однозв'язні списки об'єктів, які він зберігатиме, або покажчики на них. У змішаних таблицях індекс масиву обчислюється за допомогою *функції підставлення (hash function)*, яка приймає його ключове значення фізичного масиву як параметр. При цьому функція підставлення має гарантувати, що повернений нею індекс знаходитиметься в діапазоні фіксованих меж масиву таблиці. Вона, окрім того, повинна обчислювати індекси так, щоб вони були різними для різних ключів.

У ідеалі функція підставлення ніколи не повинна повертати один і той самий індекс для двох різних ключових значень. Проте, на практиці цей ідеал досягти дуже важко (найчастіше – неможливо). Часто буває так, що функція повертає однакові значення індексу для декількох ключів, внаслідок чого відбувається їх *зіткнення*: кожен з ключів хоче зайняти одну і ту ж саму позицію в таблиці, але місце є тільки для одного. Існують два критерії, які відрізняють хороші функції підставлення від поганих. Критерій *швидкості* вказує на те, що функції підставлення мають бути дуже швидкими. Інший критерій забезпечує *рівномірний розподіл* індексів по масиву, що знижує імовірність їх зіткнення. Для пришвидшення пошуку часто використовують лівосторонні логнормальні чи будь-які інші розподіли, які забезпечують скупчення об'єктів у змішаній таблиці ближче до її початку.

Якщо зважити на появу реальних обставин, то потрібно погодитися з тим, що зіткнення ключів все ж таки інколи траплятимуться. Важливою особливістю проектування змішаних таблиць є те, що вони мають достойно справлятися з питанням зіткнення ключів. Найчастіше зіткнення контролюють шляхом *зчеплення*, тобто кожен об'єкт, який претендує на один і той самий індекс, має приєднатися до однозв'язного списку елементів. У цьому списку внесені об'єкти зазвичай не розташовуються в якомусь певному по-

рядку: додані останніми вони можуть знаходитися на його початку. Проте разом з елементом даних у кожному об'єкті має обов'язково зберігатися його асоційований ключ.

Для здійснення пошуку об'єкта за певним його ключем, насамперед, застосовують функцію підставлення, яка обчислює індекс масиву таблиці, а потім виконують перегляд відповідного однозв'язного списку на предмет збігання ключів. Знаходження у об'єкті разом з даними і самого ключа дає змогу зберігати повторювані дані, але самі ключі при цьому повинні бути унікальними. Змішані таблиці не розуміють таких відношень, як "менше", і немає сенсу говорити про "попереднє значення". Це означає, що для змішаної таблиці немає сенсу писати ітератор (покажчик на динамічні об'єкти) – аналог покажчика на елементи масиву. На рис. 1 показано, як виглядає розташування об'єктів змішаної таблиці в пам'яті комп'ютера.

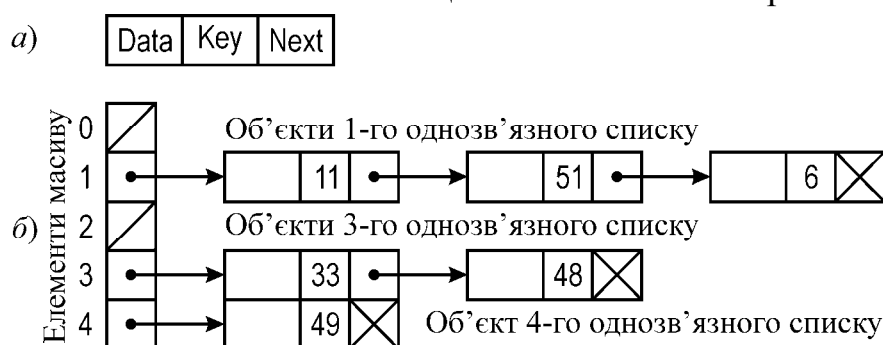


Рис. 1.
Розташування
об'єктів змішаної
таблиці в пам'яті
комп'ютера:
а) структура об'єкта;
б) структура таблиці

Розроблення *узагальненої змішаної таблиці* – не просте завдання. Для розуміння такого процесу розглянемо звичайний підхід, згідно з яким тип ключа передається у функцію підставлення Hash(), яка повертає відповідне ціле значення. Недоліком такого підходу є неможливість узагальненої змішаної таблиці обробляти вбудовані типи **int** або **char**, оскільки немає вбудованої операції Hash(). Окрім того, виникає запитання: звідки функція Hash() може знати, який допустимий діапазон індексів вона може повертати? З програмної точки зору це означає таке: якщо об'єкт в контейнері має дуже багато знати про свою реалізацію, то такий контейнер не пристосований для того, щоб бути узагальненим. Тому, зазвичай, змішані таблиці проектують окремо для ключів типу **int** і **char***.

З літератури [3, ст. 106] відомі деякі приклади реалізації змішаних таблиць, які, хоча і не є по-справжньому узагальненими, проте вони як шаблони проектування можуть достатньо широко застосовуватися у програмних продуктах. Кожна з розглянутих нижче змішаних таблиць базується на використанні специфічного типу ключа і передбачає власну функцію підставлення. Такий підхід до їх реалізації є розумним, оскільки вони часто вимагають спеціального прилаштування до розв'язання конкретних задач.

2.1. Реалізація змішаної таблиці з ключем типу **int**

Вводячи об'єкт (тобто запис, який характеризує, наприклад, елемент булевої матриці суміжності чи інцидентності) у змішану таблицю, його необхідно явно асоціювати з унікальним цілим ключем типу **int**. Як на перший погляд, то це дещо невдалий підхід. Якщо зважити на те, що змішана таблиця – це одновимірний масив фіксованого розміру (див. рис. 1, б), елементами якого

є динамічний однозв'язний список, то, напевно, для забезпечення унікальності такого ключа доведеться підтримувати ще і *другу* таблицю. Простий вихід з цієї ситуації полягає в тому, що необхідно визначити метод `AddHash()`, який буде повертати **false**, якщо ключ повторюється, і **true** – у іншому випадку.

У деяких задачах використання пошукових ключів типу `int` є дуже простим і природним представленням індексів одновимірного масиву. Відлагоджувач, наприклад, повинен уміти знаходити символічне ім'я за даною адресою або рядок початкового коду за його номером. У таких випадках розробникові програмного забезпечення не доведеться робити щось особливе, щоб генерувати ключі або забезпечити їх унікальність.

Зовсім інакша ситуація спостерігається, коли необхідно представити індекси двовимірного масиву, а ще більше питань виникає, коли це стосується багатовимірних масивів. Проте і у цих випадках є прості та ефективні рішення. Наприклад, у двовимірній матриці розміром $m \times n$ індекси будь-якого її елементу (i -го рядка і j -го стовпця) можна представити через k -ий індекс елементу одновимірного масиву. Перетворення двовимірних індексів у одновимірні може здійснюватися двома способами: $k = (i - 1) \cdot n + j$ – по рядках; $k = (j - 1) \cdot m + i$ – по стовпцях. Аналогічні дії необхідно виконати і для перетворення індексів елементів багатовимірних масивів у індекси елементів одновимірного масиву.

Особливої уваги заслуговує робота функції підставлення `Hash(int)`. У найпростішому випадку вона може повертати залишок від ділення свого аргументу (тобто індексу одновимірного фізичного масиву) на фіксований розмір масиву таблиці. Така функція завжди є роботоздатна, а також задовольняє вимогам швидкодії. Але чи забезпечує вона рівномірність розподілу підставлень по масиву? Залежить це, насамперед, від значень самих ключів. Якщо ключі розподілено випадково, то ця функція не гірша за будь-яку іншу. Але обставини іноді складаються не на користь випадкового їх розподілу.

Наприклад, у багатьох фізичних масивах адреси їх елементів з великою імовірністю можуть бути кратні чотирьом. Якщо індекси таких масивів використовуються як ключі, а розмір масиву таблиці кратний чотирьом, то багато елементів масиву таблиці можуть залишитися порожніми (див. рис. 1, б), що – поза сумнівом, поганий результат. Цю ситуацію можна звести нанівець, примусивши функцію `Hash()` зміщувати свій аргумент управо на 2 біти перед обчисленням залишку від ділення. Але можуть існувати і інші чинники, які перешкоджатимуть випадковій поведінці ключів. Загалом, вибір розміру масиву таблиці як простого числа допомагає мінімізувати поведінку подібних чинників. Для цього необхідно модифікували параметризований конструктор `HashTable()` так, щоб він використовував для задавання розміру масиву таблиці не свій аргумент, а найменше просте число, яке є більшим або дорівнює йому. Проведене нами тестування і уточнення функції підставлення і розміру таблиці принесло деякий виграш майже для кожної досліджуваної змішаної таблиці.

2.2. Реалізація змішаної таблиці з ключем типу `char *`

Додаючи об'єкт у таблицю, необхідно його явно асоціювати з унікальним рядковим ключем типу `char *`. Для деяких програмних продуктів, які пра-

цюють з таблицею символів (наприклад, листами середовища Excel), це є надзвичайно корисним механізмом. Та й збереження елементів, наприклад, матриці інцидентності не через їх індекси, а за допомогою символічних координат дає змогу вилучити операцію перетворення двовимірних індексів у одновимірні. Загалом, таблиця символів надає програмі спосіб пошуку інформації про символ, наприклад, адресу, за його рядковим представленням. Змішана таблиця з ключем типу **char*** – природний спосіб реалізації як динамічної структури даних, так і механізму пошуку відповідного об'єкта. Іншою особливістю такої змішаної таблиці є масив елементів, розмір якого може збільшуватися у міру додавання об'єктів. Коли середня довжина однозв'язних списків стає надмірно великою, то розмір масиву таблиці може збільшуватися дискретно за деякою закономірністю. У роботі [3, ст. 111] зазначено, що вибір найменшого простого числа, більшого від подвоєного попереднього розміру, значно покращує ефективність роботи змішаної таблиці.

Програмна реалізація змішаної таблиці припускає, що рядкові ключі залишаються стабільними, тобто покажчики посилатимуться на одні і ті ж незмінні рядки протягом всієї тривалості існування таблиці. Якщо таке припущення є сумнівним, то таблицю потрібно модифікувати так, щоб вона робила власну копію для кожного ключового рядка і видаляла б його під час видалення відповідного об'єкта або знищення таблиці.

2.3. Особливості програмної реалізації змішаних таблиць

З використанням об'єктно-орієнтованого підходу ми розробили проект, який реалізує динамічну структуру даних – змішані таблиці для роботи з булевими матрицями великих розмірів, пошуковими ключами яких є типи **int** і **char**. Проект складається з таких файлів: інтерфейс – у заголовному (*.h); реалізації – у початковому (*.cpp); основної функції. У заголовних файлах містяться шаблони змішаних таблиць з відповідними пошуковими ключами. Дворівневе клієнтське меню, яке знаходиться у основному файлі, дає змогу користувачу вибрати спеціальний об'єкт (запис) типу **tType**, який зразу ж заноситься у відповідний параметризований конструктор. У разі, коли метод **FindHash()** намагається відшукати відсутній у таблиці об'єкт, то цей тип слугує як індикатор "не знайдено". Вибраний користувачем тип **tType** підтримує процедуру копіювання об'єктів. Користувач, за бажанням, також може не вказувати розмір масиву таблиці, який за замовчуванням визначається автоматично.

Метод **AddHash()** виявляє у об'єктах ключі, які повторюються, і відмовляється поміщати їх у таблицю. Це означає, що користувач, перед тим, як вносити об'єкт, повинен ретельно визначити його ключ. Проте багатократне повторення даних у об'єктах допускається.

3. Приклади створення змішаних таблиць

Розглянемо процес створення змішаних таблиць з різними типами ключів на деякому конкретному навчальному прикладі. Хоча не завжди такі приклади відповідають реальним інженерним розрахункам, проте ми не намагатимемося спростувати це твердження, позаяк програма здатна працювати з надзвичайно великими масивами.

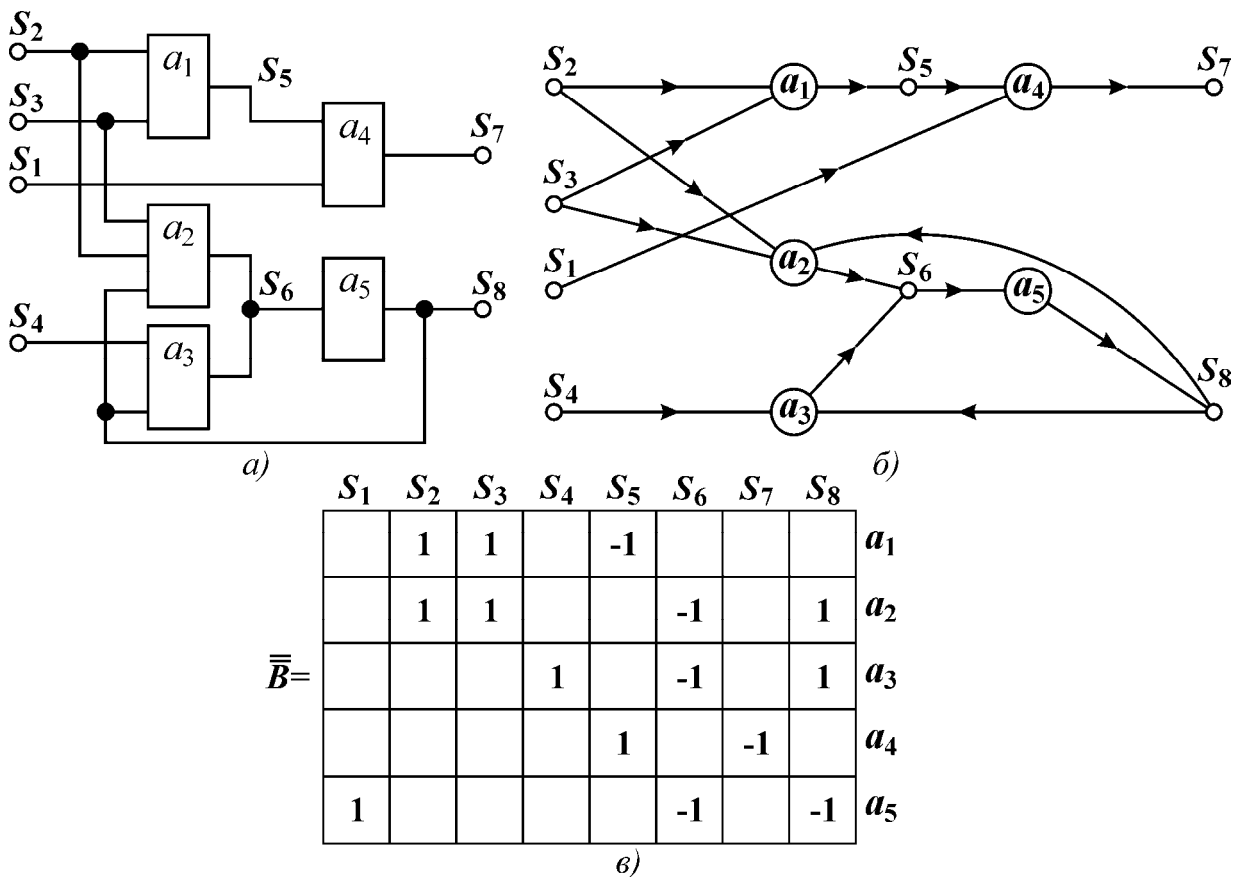


Рис. 2. Функціонально-логічне представлення цифрових пристроїв:
 а – схема логічного пристрою; б – дводольний орієнтований граф (біограф);
 в – матриця інцидентності елементів

Отож, під час функціонально-логічного проектування [1, ст. 33] моделю цифрових пристроїв слугують дводольні орієнтовані графи (біографи) з двома підмножинами вершин (рис. 2, а), які не перетинаються: підмножина V_1 утворена елементами (на більш глибокому рівні структурування) і їх з'єднаннями; підмножина V_2 відповідає сигналам – логічним змінним. Ребра такого графу відображають причинно-наслідкові зв'язки між вершинами вказаних підмножин, які відповідають статичному (підмножина V_1) і динамічному (підмножина V_2) описам об'єктів. Різниця між входами і виходами встановлюється за допомогою напрямків, прописаних ребрами: вихідний сигнал логічного елемента виходить із відповідної вершини, а вхідний сигнал має напрямок до вершини (рис. 2, б). Наведений біограф можна описати булевою матрицею $\bar{B}$ (рис. 2, в), яка відображає відношення інцидентності елементів $\bar{A} = |a_1, a_2, \dots, a_m|$ і відповідних сигналів $\bar{S} = |S_1, S_2, \dots, S_n|$:

$$\bar{B} = [\bar{A}^T \times \bar{S}] \Rightarrow b_{ij} = a_i^T \times S_j, j = \overline{1, n}; i = \overline{1, m},$$

де m і n – відповідно кількість рядків і стовпців матриці. Деякі елементи b_{ij} матриці дорівнюють $+1$, якщо S_j є вхідним сигналом елемента a_i і дорівнюють -1 , якщо S_j – вихідний сигнал елемента a_i . Усі інші елементи можуть бути нульовими або не мати значень взагалі.

На рівні кількісного проектування кожному елементу булевої матриці або логічній змінній ставиться у відповідність алгебраїчна та інша кількісна

змінна, а логічні відношення переходять у кількісні, наприклад, рівняння чи нерівності. На кожному іншому рівні проектування можливі описи об'єктів з різною ступінню повноти і узагальнення, оскільки існують різні ступені деталізації структурних, логічних і кількісних властивостей і відношень. У такій задачі значні труднощі виникають під час побудови необхідної наближеної моделі, яка б достатньо точно відображала характерні властивості об'єктів або їх елементів на тому чи іншому рівні проектування і водночас була б доступною для проведення дослідження.

Наше завдання полягає в тому, щоби за допомогою розробленого програмного проекту представити матрицю інцидентності елементів (рис. 2, в), тобто реалізувати змішану таблицю з ключами типу **int** та **char ***. Оскільки біограф, зображений на рис. 2, б, описано матрицею $\overline{B}$, тобто двовимірним масивом, то для реалізації змішаної таблиці з ключем типу **int** необхідно попередньо здійснити перетворення значень індексів двовимірного масиву у значення індексів одновимірного масиву. Це можна здійснити як по рядках, так і стовпцях (табл.), тобто результати роботи змішаної таблиці мають бути різними.

Табл. Перетворення значень індексів двовимірного масиву $\overline{B}$ у індекси одновимірного масиву при $m = 5$ і $n = 8$ (всього $5 \cdot 8 = 40$ бінарних елементів)

№ з/п	Значення індексів двовимірного масиву				Значення індексів одновимірного масиву відповідно:		Значення елемента
	i	a_i	j	S_j	$k^* = (i - 1) \cdot n + j$	$k^{**} = (j - 1) \cdot m + i$	
1	5	a_5	1	S_1	33	5	1
2	1	a_1	2	S_2	2	6	1
3	2	a_2	2	S_2	10	7	1
4	1	a_1	3	S_3	3	11	1
5	2	a_2	3	S_3	11	12	1
6	3	a_3	4	S_4	20	18	1
7	1	a_1	6	S_6	6	26	-1
8	4	a_4	6	S_6	30	29	1
9	2	a_2	6	S_6	14	27	-1
10	3	a_3	6	S_6	22	28	-1
11	5	a_5	6	S_6	38	30	-1
12	4	a_4	7	S_7	31	34	-1
13	2	a_2	8	S_8	16	37	1
14	3	a_3	8	S_8	24	38	1
15	5	a_5	8	S_8	40	40	-1

Примітка: * – по рядках; ** – по стовпцях.

З урахуванням відповідних перетворень, результати роботи розробленого програмного проекту, який реалізує змішану таблицю з ключем типу **int**, має такий вигляд:

slot #1 – перетворення двовимірного масиву у одновимірний по рядках

0: [1,24] [1,30] [-1,6] [1,3] [1,33]
 1: [-1,40] [1,16] [-1,31] [-1,22] [1,10]
 2: [-1,38] [-1,14] [1,20] [1,11] [1,2]

slot #2 – перетворення двовимірного масиву у одновимірний по стовпцях

0: [-1,30] [-1,27] [1,24] [-1,21] [1,18] [1,12] [1,6]

1: [-1,40] [1,37] [-1,34] [-1,28] [1,7]

2: [1,38] [1,11] [1,5]

Як бачимо, заповнення об'єктами змішаної таблиці у першому випадку (slot #1) є більш рівномірним, ніж у другому (slot #2), що свідчить про деякі недоліки роботи такої таблиці.

Результат роботи розробленого програмного проекту, який реалізує змішану таблицю з ключем типу **char ***, має такий вигляд

slot #3 – занесення значень елементів двовимірного масиву по рядках

0: [-1, S6-a5] [-1, S7-a4] [1, S2-a1] [1, S4-a3] [1, S8-a3]

1: [1, S2-a2] [1, S3-a1] [-1, S6-a2]

2: [-1, S8-a5] [1, S5-a4] [1, **S3-a2**] [-1, S6-a3]

3: [1, S1-a5] [-1, S5-a1] [1, S8-a2]

slot #4 – занесення значень елементів двовимірного масиву по стовпцях

0: [1, S8-a3] [-1, **S7-a4**] [-1, S6-a5] [1, S2-a1] [1, S4-a3]

1: [1, S3-a1] [1, S2-a2] [-1, S6-a2]

2: [1, S8-a5] [1, S3-a2] [1, S5-a4] [-1, S6-a3]

3: [1, S8-a2] [1, S1-a5] [-1, S5-a1]

Як бачимо, заповнення об'єктами змішаної таблиці у кожному з випадків є більш-менш рівномірним, що свідчить про деяку перевагу використання ключів типу **char *** над ключами типу **int**. Жирними шрифтом показано місцезнаходження відповідного елемента з табл.

Висновки

1. Розглянуто питання застосування динамічної структури даних – *змішаних таблиць* (hash tables) для роботи з великими масивами, значна більшість елементів яких не використовується або мають фіксовані значення.
2. З'ясовано, змішані таблиці мають здатність як компактно зберігати відповідні об'єкти, так і здійснювати ефективний доступ до них, що робить їх надзвичайно корисними в багатьох ситуаціях, які виникають під час розроблення сучасних програмних продуктів.
3. Розглянуто приклади зберігання об'єктів, які представляють елементи матриці інцидентності цифрового пристрою, у змішаних таблицях з ключами різного типу, що свідчить про ефективність роботи розробленого програмного проекту.
4. Встановлено, завдяки використанню змішаних таблиць пам'ять для зберігання кожного з об'єктів (елементів бінарної матриці) виділяється з пули вільної пам'яті комп'ютера тільки в міру потреби.

Література

1. Молчанов А.А. Моделирование и проектирование сложных систем. – К.: Выща шк. Головное изд-во, 1988. – 360 с.
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. – СПб.: Питер, 2001. – 460 с.
3. Фридман А., Кландер Л., Михаэлис М., Шильт Х. С/С++. Алгоритмы и приемы программирования. – М.: ООО "Бином-Пресс", 2007. – 560 с.