ЛЕКЦІЯ 9.

МАСИВИ, МНОЖИНИ,КОРТЕЖІ

9.1. Масиви

Масив - це набір однотипних елементів даних, з кожним з яких пов'язана впорядкована послідовність цілих чисел, які називають індексами. Індекси однозначно визначають місце даного елемента в масиві і забезпечують прямий доступ до нього. Локалізувати елемент умасиві означає задати його індекс. Кількість індексів визначає розмірністьмасиву. Кількість елементів визначають розмір масиву.

Розмір і розмірність масиву фіксовані. Найпростіший масив одномірний, його називають також вектором. Двомірниймасив - матриця, де кожний елемент *b[i,j]* належить одночасно двом лінійним ортогональним послідовностям: j-му стовпцю та *i*-му рядку. Подібну структуру мають також масиви більшої розмірності, тобто кожний індекс масиву визначає одну ортогональну лінійну послідовність так само, як координати точки в багатомірній системі координат.

Універсальні мови програмування (такі як Фортран, Паскаль, С, С++) дають можливість працювати тільки з масивами ортогональної структури. Але ці самі двомірні масиви можуть бути симетричними, діагональними, квазідіагональними, де відмінні від нуля тільки ті елементи b_{ij}, для яких j=*i* або $j = i+1$. Трикутні матриці називають тетраедральними. Крім того, існують ще так звані розріджені матриці, в яких багато елементів є нульовими. Такі структури можуть мати масиви різних розмірностей. Кожна з цих структур має свою структуру зберігання в пам'яті комп'ютера. Як правило, ортогональні багатомірні масиви відображуються у одномірні.

Найбільш поширеними операціями над структурами масивів є:

1) пошук елемента за заданим індексом;

2) локалізація елемента в масиві;

3) запис елемента в масив;

4) злиття масивів і розбиття масиву на частини;

5) сортування елементів масиву за деякими правилами;

6) копіювання масивів.

Найскладнішою є операція локалізації або обчислення індексу елемента по відношенню до першого елемента. Розглянемо модифікації цієї операції для різних типів масивів і способів відображення їх у одномірний масив.

Очевидно, що локалізація елемента вектора не представляє ніяких труднощів, оскільки індекс елемента однозначно визначається його номером у послідовності.

Індекс елемента b_{ij} двомірного масиву, що складається із *n* рядків та *m* стовпців, при зберіганні в пам'яті комп'ютера "по стовпцях", обчислюється за формулою ***(j-1)*n + (i-1)*** , а при зберіганні його "по рядках" - за формулою ***(i-1)*m+(j-1).***

У загальному випадку для *n* -мірногомасиву, елемент якого позначається $B[S_1, \dots, S_n]$, а діапазон зміни індексів визначається нерівностями $1 <= S_i <= u_i$,$i = 1,n$, функція адресації цього масиву при запам'ятовуванні його "по рядках" має вигляд

$$f(S_1, \dots, S_n) = u_2\ u_3 \dots\ u_n (S_1-1) + u_3\ u_4 \dots\ u_n (S_2-1) + \dots + u_n (S_{n-1}-1) + (S_n-1).$$

Оперативна пам'ять із погляду програміста - це масив елементів. Будь-який елемент масиву

можна прочитати або записати відразу, за одну елементарну дію. Масив можна розглядати як базову структуру даних. Структури даних, у яких можливий безпосередній доступ до довільних їхніх елементів, називають структурами даних із **прямим**, або з**довільним доступом** (англійською random access).

З логічної точки зору, масивом є також найважливіша складова комп'ютера - магнітний диск. Елементарною одиницею читання й запису для магнітного диска служить блок. Розмір блоку залежить від конструкції конкретного диска, звичайно він кратний 512. За одну елементарну операцію можна прочитати або записати один блок із заданою адресою.

Отже, найбільш важливі запам'ятовувальні пристрої комп'ютера - оперативна пам'ять і магнітний диск - являють собою масиви. Робота з елементами масиву здійснюється винятково швидко, усі елементи масиву доступні без усяких попередніх дій. Протемасивів недостатньо для написання ефективних програм. Наприклад, **пошук елемента** в масиві, якщо його елементи не впорядковані, неможливо реалізувати ефективно: не можна винайти нічого кращого, крім послідовного перебору елементів. У випадку впорядкованого зберігання елементів можна використовувати ефективний **бінарний пошук**, але утруднення виникають при додаванні або видаленні елементів у середині масиву й приводять до **масових операцій**, тобто операцій, час виконання яких залежить від числа елементів структури. Від цих недоліків вдається позбутися, реалізуючи множину елементів на базі **збалансованих дерев** або **хеш-функцій**.

Є й інші причини, за яких необхідно використовувати більш складні, ніжмасиви, структури даних. Логіка багатьох задач вимагає організації певного порядку доступу до даних. Наприклад, у випадку**черги** елементи можна додавати тільки в кінець, а забирати тільки з початку черги; у **стеці** доступні лише елементи у вершиністека, у **списку** - елементи до й за вказівником.

Нарешті, масив має обмежений розмір. Збільшення розміру масиву приводить до переписування його вмісту в захоплену область пам'яті більшого розміру, тобто знову ж до масової операції. Від цього недоліку вільні **посилальні реалізації**структур даних: реалізації на основі лінійних списків або на основі дерев.

9.2. Множини і кортежі

Множина - найпростіша структура, в якій між окремими ізольованими елементами немає ніякого внутрішнього зв'язку. Набір таких елементів являє собою множину, яка не має ніякої структури. Це сукупність даних деякого типу, елементи якої мають певну властивість. Але повинні бути чітко встановлені область визначення елементів даних і правила їх відбору у множину. Основними операціями над множинами є об'єднання, перетин і різниця. Множину, на якій встановлено відношення порядку " <= ", називають впорядкованою. Якщо таке відношення має місце для всіх елементів множини, таку множину називають повністю впорядкованою, інакше - це частково впорядкованамножина. Множину *M* називають індексованою, якщо задане її відображення в натуральний ряд чисел, тобто *I : M à{1,2,....|x|],* де | *x*| - потужність множини*M.* У множину *M* можна додати елемент*x*, з множини *M* можна видалити елемент*x*. Якщо при додаванні елемента *x* він уже міститься в множині *M*, то нічого не відбувається. Аналогічно, ніякі дії не відбуваються при видаленні елемента *x*, коли він не міститься в множині *M*. Нарешті, для заданого елемента *x* можна визначити, чи належить він множині *M*.Множина - це потенційно необмежена структура, вона може містити будь-яке кінцеве число елементів.

У деяких мовах програмування накладають обмеження на тип елементів і на максимальну кількість елементів множини. Так, іноді розглядають множину елементів дискретного типу, число елементів якого не може перевищувати деякої константи, що задається при утворенні множини. (Тип називається дискретним, якщо всі можливі значення даного типу можна занумерувати цілими числами.) Для таких множин вживають назву *Bitset* ("набір бітів") або просто *Set*. Як правило, для реалізації таких множин використовується бітова реалізація множини на базі масиву цілих чисел. Кожне ціле число розглядається у двійковому представленні як набір бітів, що містить 32 елемента. Біти усередині одного числа нумеруються справа наліво (від молодших розрядів до старших); нумерація бітів триває від одного числа до

іншого, коли ми перебираємо елементимасиву. Наприклад, масив з десяти цілих чисел містить 320 бітів, номери яких змінюються від 0 до 319. Множина у даній реалізації може містити будь-який набір цілих чисел у діапазоні від 0 до 319. Число*N* належить множині тоді й тільки тоді, коли біт з номером *N* дорівнює одиниці. Відповідно, якщо число *N* не належить множині, то біт з номером *N* дорівнює нулю. Нехай, наприклад, множина містить елементи 0, 1, 5, 34. Тоді в першому елементі масиву встановлені біти з номерами 0, 1, 5, у другому - біт з номером 2 = 34 - 32. Відповідно, двійкове представлення першого елемента масиву рівно 10011 (біти нумеруються справа наліво), другого - 100, це числа 19 і 4 у десятковому представленні. Усі інші елементи масиву нульові.

У програмуванні досить часто розглядають структуру більш складну, чим проста множина: **навантажена множина**. Нехай кожний елемент множини міститься в ній разом з додатковою інформацією, яку називають навантаженням елемента. При додаванні елемента в множину потрібно також вказувати навантаження, яке він несе. У різних мовах програмування й у різних стандартних бібліотеках такі структури називають **Картою** (Map) або**Словником** (Dictionary). Дійсно, елементи множини як би наносяться на навантаження, яке вони несуть. В інтерпретації Словника елемент множини - це іноземне слово, навантаження елемента - це переклад слова на певну мову (зрозуміло, переклад може включати кілька варіантів, але тут переклад розглядається як єдиний текст).

Всі елементи містяться в навантаженій множині в одному екземплярі, тобто різні елементи множини не можуть бути рівні один одному. На відміну від самих елементів, їх навантаження можуть збігатися (так, різні іноземні слова можуть мати однаковий переклад). Тому елементи навантаженої множини називають ключами , їх навантаження - значеннями ключів. Кожний ключ унікальний. Прийнято говорити , що ключі відображаються на їхні значення. Як приклад навантаженої множини поряд зі словником можна розглянути множину банківських рахунків. Банківський рахунок - це унікальний ідентифікатор, що складається з десяткових цифр. Навантаження рахунку - це вся інформація, яка йому відповідає, що включає ім'я й адресу власника рахунку, код валюти, суму залишку, інформацію про останні транзакції й т.п.

Найбільш часто застосовувана операція в навантаженій множині - це визначення навантаження для заданого елемента x(значення ключа x). Реалізація цієї операції включає пошук елемента x у множині, тому ефективність будь-якої реалізації множини визначається насамперед швидкістю пошуку.

Кортеж - елемент n - кратного добутку множини $X : X*X*..*X=X^n$. На відміну від скінченної впорядкованої множини, яка є підмножиною декартового добутку деяких множин $X_1, X_2, ..., X_n$ елементи кортежу можуть повторюватись.

9.2.1. Реалізація множини

Розглянемо для простоти реалізацію звичайної, не навантаженої множини. (Реалізація навантаженої множини аналогічна реалізації звичайної, просто паралельно з елементами потрібно додатково зберігати навантаження елементів.) У звичайній реалізації множини її елементи зберігаються вмасиві, починаючи з першої комірки. Спеціальна змінна містить поточне число елементів множини, тобто кількість використовуваних у цей момент комірокмасиву.

При додаванні елемента x до множини спочатку необхідно визначити, чи міститься він у множині (якщо міститься, то множина не змінюється). Для цього використовується процедура пошуку елемента, яка відповідає на запитання, чи належить елемент x множині, і, якщо належить, видає індекс комірки масиву, що містить x. Та ж процедура пошуку використовується й при видаленні елемента з множини. При додаванні елемента він дописується в кінець (у комірку масиву з індексом рівним числу елементів) і змінна числа елементів збільшується на одиницю. Для видалення елемента достатньо останній елемент множини переписати на місце, що видаляється й зменшити змінну числа елементів на одиницю.

У звичайній реалізації елементи множини зберігаються в масиві в довільному порядку. Це означає, що при пошуку елемента x доведеться послідовно перебрати всі елементи, поки ми або

не знайдемо *x*, або не переконаємося, що його там немає. Нехай множина містить 1.00.00 елементів. Тоді, якщо *x* належить множині, доведеться переглянути в середньому 500.00 елементів, якщо немає - усі елементи. Тому звичайна реалізація підходить тільки для невеликих множин.

9.3. Зберігання множин і масивів

Множини це структури найпростішого вигляду. В пам‘яті їх можна зображувати двома способами:

1) для кожного елемента множини зберігати в пам‘яті його описання аналогічно до математичного способу задання множини переліком її елементів;

2) визначити всі потенційно можливі елементи множини, а потім для всякої підмножини такої універсальної множини вказувати для кожного можливого елемента, чи належить він даній підмножині, чи ні. Цей спосіб аналогічний предикатній формі задання множини в математиці.

У першому випадку множину зручно зберігати у вигляді вектора або лінійного списку. У другому - використовуються вектори бітів. Для цього пронумеруємо всі можливі елементи множини, наприклад, від 1 до *N* . Відведемо вектор пам‘яті із *M* бітів і визначимо множину, встановлюючи в *i* -й біт одиницю, якщо *i*-й можливий елемент присутній у даній множині, і нуль - у протилежному випадку. Очевидно, якщо *N* велике, а конкретна підмножина мала, то краще користуватися першим способом. Другий спосіб має переваги, які дозволяють виконувати операції над множинами за допомогою логічних операцій машинного рівня.

Слід відзначити, що як тільки множинавиявиться записаною в пам‘яті, спосіб її зображення визначає на ній індексацію, і різні методи зображення дозволяють по-різному її обробляти. Зображення в пам‘ятімасивів очевидне. Найпростіший масиводномірний, який ототожнюється з вектором. Багатомірні масиви також зображуються векторами. Формули для локалізації елементів багатомірного масиву у векторному зображенні наведено в підрозд.6.1. Однак так звані розріджені матриці зображуються в пам‘яті нелінійними ба-гатозв‘язними структурами.

9.4. Зберігання розріджених матриць

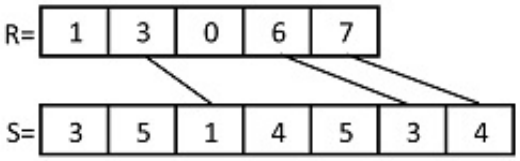
Розрідженими називають такі матриці, більшість елементів яких дорівнює нулю. Такі матриці зберігаються у вигляді зв‘язаних структур. Існують спеціальніалгоритми введення і множення розріджених матриць, зображених у такій формі. Одним з основних способів їх зберігання є табличний. Він полягає у запам'ятовуванні ненульових елементів матриці в одномірному масиві та ідентифікації кожного елемента масиву індексами рядка і стовпця.

Наприклад, нехай задана матриця:

$$A=\begin{Vmatrix} 0 & 0 & 6 & 0 & 9 \\ 2 & 0 & 0 & 7 & 8 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 12 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{Vmatrix}$$

A[1,3]=6 A[1,5]=9
A[2,1]=1 A[2,4]=7
A[2,5]=8 A[4,3]=12
A[5,4]=3

Її можна зберігати у вигляді трьох векторів, які містять відповідно ненульові елементи матриці, а також індекси їх рядків та індекси стовпців: *Z={6,9,2,7,8,12,3}* - значення ненульових елементів; *R ={1,1,2,2,2,4,5}* - індекси рядків; *S={3,5,1,4,5,3,4}* - індекси стовпців. Щоб не було повторень у векторі індексів рядків R , у ньому можна зберігати тільки індекс першого елемента у послідовності S:


R=
1
3
0
6
7
S=
3
5
1
4
5
3
4