УДК 004.942(045)

[1]**В. М. Синєглазов**, д.т.н., проф.
[2]**С. Г. Кєворков,** асп.
[3]**Раад Карім Кадем,** асп.

ВИБІР ОПТИМАЛЬНОГО АЛГОРИТМУ ПОШУКУ МОДУЛЯ ІНТЕГРАЦІЇ ПРОЕКТНИХ ДАНИХ СИСТЕМИ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ

Національний авіаційний університет
[1]E-mail: svm@nau.edu.ua
[2]E-mail: kyevorkov@ukr.net
[3]E-mail: svm@nau.edu.ua

Визначено критерії оптимальності для вибору алгоритму пошуку, використаного у модулі інтеграції проектних даних.

Researches of forming of CAD system metamodels and its components design are conducted.

Определены критерии оптимальности для выбора алгоритма поиска, используемого в модуле интеграции проектных данных.

Постановка проблеми

Система автоматизованого проектування (САПР) у ГОСТ 23501.0-79 визначена як організаційно-технічна система, що складається з комплексу засобів автоматизації проектування, який взаємодіє з підрозділами проектної організації та виконує автоматизоване проектування.

Засоби автоматизації проектування можна згрупувати за видами забезпечення автоматизованого проектування.

Технічне забезпечення САПР є сукупністю взаємозалежних і взаємодіючих технічних засобів, призначених для виконання автоматизованого проектування.

Технічне забезпечення поділяється на групи засобів програмного оброблення даних, підготування та введення даних, відображення й документування, архівації проектних рішень, передавання даних.

Засоби програмного оброблення даних подано процесорами й запам'ятовувальними пристроями, тобто пристроями ЕОМ, в яких реалізуються перетворення даних і програмне керування обчисленнями.

Засоби підготування, введення, відображення й документування даних служать для спілкування людини з ЕОМ. Засоби архіву проектних рішень подано зовнішніми запам’ятовувальними пристроями.

Засоби передавання даних використовуються для організації зв’язків між територіально рознесеними ЕОМ і терміналами (кінцевими пунктами).

Існуючі підсистеми проектування у своїй більшості є інформаційно-пошуковими або типовими варіантними.

Методи автоматизації проектних робіт підрозділяють на три групи.

До першої групи методів автоматизації ставляться інформаційні, пошукові підсистеми.

Друга група методів автоматизації є типовим варіантним проектуванням, за якого створюється узагальнена структура. Потім з неї синтезується конкретна структура або структури. У більшому або меншому ступені друга група використовується у разі типізації. Типове варіантне проектування застосовує різні оператори проектування, засновані на наборі евристичних методів, логічних або математичних алгоритмів.

Третя група методів автоматизації включає творчі дії, спрямовані на невідоме. Зокрема, застосовують евристичне програмування.

Велику увагу приділяють підвищенню ефективності САПР для повнішого використання даних (у цифровому й символьному

форматі) під час вирішення завдань, прямо не пов'язаних із проектуванням, наприклад, завдань технологічної підготовки виробництва (ТПВ).

У роботі [1] запропоновано підхід до побудови цілеорієнтованої САПР, спрямованої на вирішення завдань ТПВ.

Відмінна риса такої САПР – наявність модуля інтеграції проектних даних, основними елементами якого є:

– база даних;

– адаптери до CAD систем;

– інтерфейси до зовнішніх систем;

– інтерфейси кінцевих користувачів;

– засоби адміністрування та кастоміизації даних;

– алгоритми сортування, консолідації, пошуку, конвертації й перетворення даних.

Пошук входить у перелік завдань, необхідних для виконання ТПВ і забезпечення вибору необхідних даних САПР.

Мета роботи – аналіз алгоритмів пошуку для застосування їх у модулі інтеграції проектних даних.

Алгоритми пошуку розділяють на статичні й динамічні [1, с. 247–273].

У разі статичного пошуку масив значень не змінюється під час роботи алгоритму. Під час динамічного пошуку масив може перебудовуватися або змінювати розмірність.

Алгоритми пошуку можна розділити на такі:

– алгоритми, що використовують істинні ключі;

– алгоритми, що працюють за перетвореними ключами.

Ключем називають те значення, яке шукаємо.

Крім того, існують алгоритми пошуку, засновані на порівнянні самих значень, та алгоритми, засновані на їх цифрових властивостях [2, с. 20–36] – алгоритми хешування.

Від вибору оптимального алгоритму пошуку залежить ефективність роботи модуля інтеграції проектних даних.

Постановка завдання

Текст, що складається з n символів, назвемо T, а $T[i]$ – його i-й символ. Рядок або слово, що складається з m символів, назвемо S, де $S[i]$ – i-й символ рядка. Потрібно перевірити, чи входить цей рядок у заданий текст, і, якщо входить, то починаючи з якого символу тексту.

Необхідно вибрати алгоритм пошуку, оптимальний за мінімумом критерію часу пошуку, з урахуванням обробки масивів різнорідних даних, що містять різну кількість елементів, для його наступного використання в модулі інтеграції проектних даних.

Найпростіший алгоритм

Суть найпростішого алгоритму полягає в такому [4]: перевіряємо, чи збігаються m символів тексту (починаючи з обраного) із символами нашого рядка, намагаючись приміряти шаблон куди можливо. Час роботи всього алгоритму є $O((n - m+1)*m)$. Для маленьких рядків пошук приходить швидко.

Основний недолік найпростішого алгоритму полягає в тому, що доводиться виконувати багато зайвої роботи.

Алгоритм пошуку за бінарним деревом

Дія алгоритму пошуку за бінарним деревом являє собою шлях пошуку у вигляді дерева, у якого кожна наступна гілка розділяється на дві, по одній із яких рухаємося далі.

Цей алгоритм має високу ефективність разом із простотою програмування. Саме бінарний пошук використовується для пошуку в індексах таблиць.

Пошук за деревом Фібоначчі

Ефективність алгоритму за деревом Фібоначчі трохи вища, ніж в алгоритмі пошуку за бінарним деревом, але пропорційна $\mathrm{Log}(2)N$.

У дереві Фібоначчі числа в дочірніх вузлах відрізняються від числа в батьківському вузлі на одну й ту саму величину, а саме на число Фібоначчі. Суть алгоритму в тім, що, порівнюючи шукане значення з черговим

значенням у масиві, не ділять навпіл нову зону пошуку, як у бінарному пошуку, а відступають від попереднього значення, з яким порівнювали, у потрібну сторону на число Фібоначчі [3, с. 110–115].

Алгоритм Фібоначчі більш ефективний, тому що містить у собі тільки такі арифметичні операції, як додавання й віднімання.

Необхідності в діленні на 2 немає, тому заощаджується процесорний час.

Алгоритм екстраполяцій

На відмінність від попередніх алгоритмів алгоритм екстраполяції не просто визначає зону нового пошуку, але й оцінює величину нового кроку.

Швидкість збіжності алгоритму більша, ніж швидкість алгоритму пошуку за бінарним деревом і деревом Фібоначчі.

Якщо у разі пошуку за бінарним деревом (алгоритм дихотомії) за кожний крок масив пошуку зменшувався з N значень до $N/2$, то, використовуючи алгоритм екстраполяції, за кожний крок зона пошуку зменшується з N значень до кореня квадратного з N. Якщо K лежить між Kn і Km, то наступний крок роблять від n на величину

$$(n - m)*(K - Kn)/(Km - Kn).$$

Швидкість екстраполяційного алгоритму починає істотно перевищувати швидкість алгоритму половинного розподілу за більших значень N.

При оцінці часу, що витрачає алгоритм на пошук, мають значення дві величини:

– розмірність масиву (кількість елементів);

– кількість звернень (скільки разів необхідно здійснювати пошук).

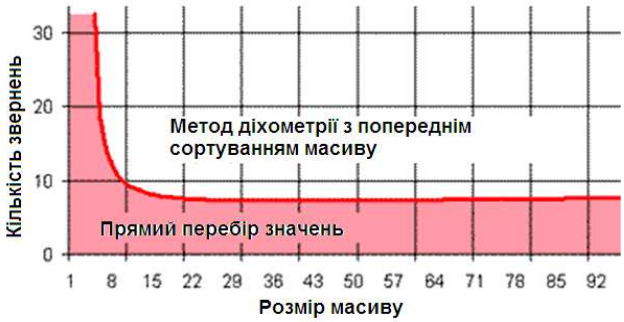
Отже, в масиві розмірністю N пошук проводимо M разів.

Кількість операцій, які виконає алгоритм прямого перебору, пропорційно $M*N$ (час $T1$). Алгоритм дихотомії здійснить $2M*\text{Log}(2)N$ операцій і додатково витрачає час на сортування – $N*\text{Log}(2)N$ (час $T2$):

$$T1 = M*N;$$

$$T2 = 2M*\text{Log}(2)N + N*\text{Log}(2)N .$$

У разі $T1 = T2$ перебуваємо на межі, на якій однаково оптимальні обидва алгоритми. Із цієї рівності можна одержати ділянки в системі координат «кількість звернень – розмірність масиву», що визначають оптимальність одного алгоритму щодо іншого (див. рисунок).



Межа використання методу прямого перебору й методу дихотомії з попереднім сортуванням масиву

Для поліпшення основних алгоритмів пошуку використовують методи корекції, які є модифікаціями того або іншого методу.

Критерієм вибору способу корекції є розмір масиву, в якому здійснюється пошук.

Конкретні умови завдання можуть істотно підвищити швидкість уже обраного алгоритму.

Алгоритм списку, що «самоорганізовується»

Ідея алгоритму списку, що самоорганізовується, така: знайдений потрібний конверт кладуть у початок пачки, тобто переміщують значення в початок масиву.

У підсумку часто використовувані елементи будуть розташовані досить близько до початку масиву. Таким чином, кількість порівнянь для вдалого пошуку буде мінімізуватися сама пособі.

Математично це можна пояснити так: припустимо, що значення Ki буде розшукуватися з імовірністю Pi, причому

$$P1 + P2 + ... Pn = 1.$$

Час удалого пошуку пропорційно C , середнє значення якого

$C = P1 + 2*P2 + ... N*Pn.$

Мінімального значення C досягне у разі

$P1 > P2 > ... > Pn,$

коли найбільш часто використовувані записи перебувають на початку таблиці.

Цей спосіб варто застосовувати у випадку найбільшої ймовірності саме вдалого пошуку. Це не самостійний алгоритм пошуку, а корекція одного з описаних алгоритмів. Під час вдалого пошуку значення відповідний запис видаляється з масиву, зменшуючи його розмірність.

Кількість порівнянь можна мінімізувати, заздалегідь відтинаючи невдалий пошук, тобто значення, яких немає в масиві. Припустимо, що є великий масив відсортованих значень. Достатньо перед початком пошуку чергового значення перевірити, а чи попадає воно в масив. Але застосовувати цей спосіб корекції алгоритму можна тільки в тому випадку, якщо ймовірність невдалого пошуку досить велика. У разі великої розмірності масиву додавати порівняння на кожному кроці можна тільки в тому випадку, якщо чергове порівняння замінить черговий виток пошуку.

Оцінимо, якою повинна бути ймовірність невдалого пошуку, тобто відсутність шуканого значення в масиві, щоб було вигідно застосовувати цю модифікацію алгоритму.

Нехай P – імовірність того, що шуканого значення немає між мінімальним і максимальним значенням масиву. Тоді, відтинаючи найбільш екстремальні значення, у середньому не робимо $P*(C - 2)$ операцій, де C – середня кількість операцій у разі проведення немодифікованого пошуку. У той же час змушені даремно зробити $2*(1 - P)$ операцій для порівняння значення з мінімальним і максимальним у масиві.

Оскільки необхідно бути у виграші, то

$P*(C - 2) > 2*(1 - P),$

$PC - 2P + 2P - 2 > 0,$

$PC > 2, P > 2/C.$

Таким чином, за $P > 2/C$ слід використовувати алгоритм списку, що самоорганізується. Наприклад, для пошуку за бінарним деревом у випадку масиву з 32 елементами досить один раз із чотирьох задавати найбільш екстремальні значення для пошуку, щоб модифікований алгоритм працював швидше.

Алгоритм Рабина-Карпа

Алгоритм Рабина-Карпа має на увазі поставити у відповідність кожному рядку деяке унікальне число і замість того, щоб порівнювати самі рядки, порівнювати числа, що набагато швидше.

Але шуканий рядок може бути довгим, рядків у тексті теж вистачає. Оскільки кожному рядку треба зіставити унікальне число, то і чисел повинно бути багато, а отже, числа будуть великими (порядку Dm, де D – кількість різних символів), і працювати з ними буде так само незручно.

Алгоритм Рабина-Карпа виконує лінійний прохід по рядку (m кроків) і лінійний прохід по всьому тексту (n кроків), тобто, загальний час роботи – $O(n + m)$. Цей час лінійно залежить від розміру рядка й тексту, і програма працює швидко.

При постановці у відповідність рядку його числове подання виникає проблема великих чисел. Її можна уникнути, якщо виконувати всі арифметичні дії за модулем якогось простого числа (постійно брати залишок від ділення на це число).

Таким чином, знаходимо не саме число, що характеризує рядок, а його залишок від ділення на якесь просте число. Тепер ставимо число у відповідність не одному рядку, а цілому класу, але оскільки класів буде досить багато (стільки, скільки різних залишків від ділення на це просте число), то додаткову перевірку виконують нечасто.

Отже, доводиться робити порівняння рядків посимвольно, але тому що «холостих» спрацьовувань буде небагато (в $1/P$ випадках), то й очікуваний час роботи малий. Час роботи – $O(m+n+mn/P)$, mn/P невеликий, так що складність роботи майже лінійна.

Просте число варто вибирати більшим. Чим більше це число, тим швидше буде працювати програма. Алгоритм Рабина-Карпа значно швидше попереднього й цілком підходить для роботи з дуже довгими рядками.

Алгоритм Кнута–Морріса–Пратта

Алгоритм Кнута–Морріса–Пратта використовує попереднє оброблення шуканого рядка, а саме: на її основі створюється префікс-функція. Суть цієї функції в знаходженні для кожного підрядка $S[1 \dots i]$ рядка S найбільшого підрядка $S[1 \dots j]$ ($j < i$), який одночасно присутній і на початку, і наприкінці підрядка (як префікс і як суфікс). Наприклад, для підрядка abcHelloabc таким під-рядком є abc (одночасно й префіксом, і су-фіксом). Зміст префікс-функції в тім, що можемо відкинути завідома невірні варіанти, тобто якщо під час пошуку збігся підрядок abcHelloabc (наступний символ не збігся), то можна продовжувати пошук уже з четвертого символу (перші три співпадуть).

Для обчислення префікс-функції враховують що, якщо префікс (він же суфікс) довгого рядка i довше одного символа, то він одночасно і префікс довгого підрядка i–1.

Таким чином, перевіряємо префікс попереднього підрядка. Якщо цей префікс не підходить, то префікс її префікса і т. д. Діючи так, знаходимо найбільший шуканий префікс.

Час роботи процедури лінійний, тому що присвоєння префікс-функції відбувається чітко m раз, решта часу міняється змінна k. Оскільки в циклі while вона зменшується ($P[k]<k$), але не стає менше 0, то зменшуватися вона може не частіше, ніж зростати. Змінна k зростає на 1 не більше m раз. Відповідно, змінна k міняється всього не більше $2m$ раз, час роботи всієї процедури $O(m)$.

Найпростіший алгоритм і алгоритм Кнута–Морріса–Пратта крім знаходження самих рядків визначають кількість збіглих символів у процесі.

Алгоритм Кнута–Морріса–Пратта дещо більш громіздкий, ніж найпростіший, але і працює набагато швидше.

Висновки

У результаті проведених досліджень алгоритмів пошуку в масивах різнорідних даних елементів, що містять різну кількість, в модулі інтеграції проектних даних рекомендується в загальному випадку використовувати алгоритм пошуку Кнута–Морріса–Пратта.

Для пошуку в невеликому масиві й нечасто пошуку (довідники покриттів або техпроцесів) можна скористатися алгоритмом перебору всіх значень масиву.

Якщо необхідно шукати серед інформаційних об’єктів деталь або складальну одиницю, то рекомендується попередньо мати збережений відсортований список елементів і згодом використати алгоритми пошуку за деревом (екстарполяційний) з відсортованим списком. З огляду на велику кількість запитів від користувачів на пошук даних САПР, попередньо збережений відсортований список істотно скоротить час пошуку.

Література

1. *Кнут Д.* Искусство программирования. Т. 3. Сортировка и поиск. – 2-е изд. / Д. Кнут (The Art of Computer Programming). Vol.3. Sorting and Searching. – М.: Вильямс, 2007. – 824 с.

2. *Аветисян Д.О.* Проблемы информационного поиска / Д.О. Аветисян. – М.: Финансы и статистика, 1981. – 206 с.

3. *Филиппова Е.* Алгоритмы сортировки. Ч.1 / Е. Филиппова. – Режим доступа: http://www.delphikingdom.com/asp/viewitem.asp?catalogid=65.

4. *Финн В.К.* Логические проблемы информационного поиска/ В.К. Финн. – М.: Наука, 1976. – 152 с.

5. *Ткачук В.* Алгоритмы поиска / В. Ткачук. – Режим доступа: http://docs.com.ru/algoritm_3.php.

Стаття надійшла до редакції 16.07.10.