МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет
«Харківський політехнічний інститут»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання самостійних робіт
з курсів: «Інформаційні технології і програмування»
«Інформаційні процеси в телекомунікаційних системах»

ІНФОРМАЦІЙНИЙ ПОШУК

для студентів спеціальностей:
151 Автоматизація та комп'ютерно-інтегровані технології
172 Телекомунікації та радіотехніка

затверджено
редакційно-видавничою
радою університету,
протокол №3 від 26.10.2022 р.

Харків
НТУ «ХПІ»
2022

Методичні вказівки для виконання самостійних робіт «Інформаційний пошук» для студентів спеціальностей 151 Автоматизація та комп'ютерно-інтегровані технології, 172 Телекомунікації та радіотехніка / Уклад. А.О. Зуєв, А.В. Івашко, Д.А. Гапон - Х.: НТУ «ХПІ», 2022. - 38 с.

Укладачі: А. О. Зуєв
А.В. Івашко
Д.А. Гапон

Рецензент О.Є. Тверитникова

Кафедра інформаційно-вимірювальних технологій і систем

ВСТУП

Інформаційний пошук - процес, в ході якого, з послідовності здійснюється вибірка даних з елементів відповідних заданому ключу. Мета будь-якого пошуку полягає в потребі і необхідності знаходити різні види інформації по заданому критерію, що сприяють отриманню, необхідних відомостей і знань для підвищення професійного і культурного рівня; створення нової інформації і формування нових знань, а також для прийняття управлінських рішень. Пошукові засоби і технології, які використовуються для реалізації інформаційних потреб, визначаються типом і розв'язуваної користувачем завданням: станом його знань про досліджуваний об'єкт. Крім того, процес взаємодії користувача з системою визначається рівнем знань користувача (повноти подачі, достовірності джерела тощо) і функціональних можливостей системи як інструменту.

Методичні вказівки призначені для вивчення основ інформаційних процесів в автоматизованих системах управління, а також алгоритмів інформаційного пошуку. Розглянуто приклади реальних систем управління і основні алгоритми пошуку інформації, а також розширений варіант пошуку з індексацією по ключу - хешування. Для всіх алгоритмів розглянуті приклади реалізації на мові С++.

Також в методичних вказівках представлені індивідуальні завдання для виконання лабораторних і практичних робіт, які допоможуть ефективному освоєнню основ інформаційного пошуку в системах управління.

1 ІНФОРМАЦІЯ ТА ІНФОРМАЦІЙНІ ПРОЦЕСИ

1.1 Інформація в контурі управління

Термін **інформація** – походить від латинського слова, що означає роз'яснення, обізнаність. Іншими словами - відомості про що-небудь, які сприймаються органами чуття людини.

Види інформації (за ступенем спадання важливості): візуальна, аудіальна, тактильна, нюхова, смакова.

За значущістю інформація буває: актуальна, достовірна, справжня, повна, зрозуміла.

Інформаційні процеси в системах управління – це процеси отримання, перетворення і передачі інформації.

Процес взаємодії між повідомленням і відправником або споживачем інформації називається інформаційним процесом. Наприклад, збір, обробка, зберігання і пошук інформації. Діяльність людини, яка пов'язана із зазначеними процесами, називають інформаційною діяльністю. Узагальнена функціональна схема інформаційного процесу в системах управління приведена на рис. 1.1.

Об'єкт управління (ОУ) і керуюча система є функціональний комплекс, який прийнято називати - система управління (СУ), в тому випадку якщо вирішуються завдання управління.

У СУ інформація в загальному випадку переміщається по замкнутому контуру. Таке переміщення відбувається за рахунок реакції на вплив (на вхідні сигнали), результатом цієї реакції є вихідні сигнали. Будь-яка система управління створюється для досягнення певного безлічі станів ОУ і реалізується шляхом вирішення конкретних завдань управління процесом перекладу ОУ з одного стану в інший.

Для вирішення завдань управління необхідна інформація про стан ОУ, пов'язана безпосередньо з розв'язуваної завданням управління - цільова інформація.

Аналіз процесу переміщення інформації в контурі управління дозволяє зробити наступні важливі висновки:

1. Рішення задач управління можливо, тільки якщо реакція об'єкта на вплив відома (може бути передбачена), що дає можливість заздалегідь розробити алгоритм управління.

2. Без наявності цільової інформації ефективне рішення відповідної задачі управління неможливо.

3. Незважаючи на імовірнісний характер, можливих зовнішніх впливів і відмов, особливістю СУ як інформаційних систем є забезпечення однозначності реакції вирішального пристрою на цільову інформацію (детермінованість алгоритму управління).

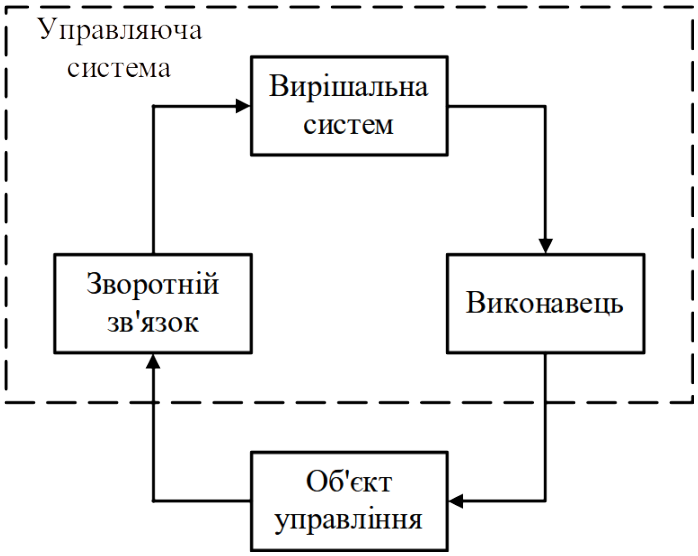

Рисунок 1.1 – Узагальнена функціональна схема інформаційного процесу

Реалізація СУ згідно з цими правилами, забезпечує:

- рішення поставленого завдання управління;
- однозначність управляючих впливів;
- передбачуваність реакції ОУ на ці дії.

Таким чином, правильно спроектована система управління є детермінованою (передбачуваною) системою.

1.2 Функціональна схема автоматизованої системи управління

На рис. 1.2 представлена функціональна схема автоматизованої системи управління (АСУ). На цій схемі видно два контури управління: контур автоматичного і контур дистанційного управління, здійснюваного оператором.

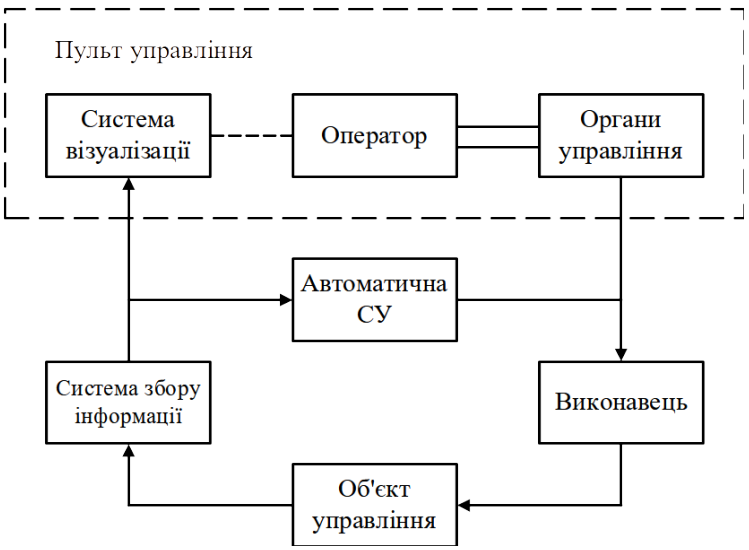

Рисунок 1.2 – Функціональна схема автоматизованої системи управління

При управлінні об'єктом можна виділити два основних потоки: інформаційний і керуючий. У керуючій системі, інформація перетворюється в команди управління за заданими алгоритмами. Особливістю контуру дистанційного управління є наявність

людини-оператора, засобів візуалізації (ЗВ) та органів управління. На ЗВ видається модель ОУ, в реальному часі відображає стан ОУ в будь-якому режимі.

Автоматична система також передає на ЗВ інформацію двох типів: про функціонування автоматики і рекомендаційну для оператора.

1.3 Ентропія в системах управління

Поняття ентропії нерозривно пов'язане з поняттям інформаційної невизначеності, яка обумовлена відсутністю або недостатньою кількістю інформації, необхідної для вирішення завдання. Отримання додаткової цільової інформації знижує ентропію системи.

ОУ є основним джерелом інформації в системі, і містить її в повному обсязі. Тому ентропія (невизначеність) ОУ як джерела інформації дорівнює нулю. Об'єкт може дати про себе будь-яку інформацію. Але, так як оператор отримує інформацію про стан ОУ з ЗВ, то величина ентропії залежить від якості ЗВ. У найпростішому випадку, можливість управління ОУ визначається співвідношенням:

$$I \leq I_{ЗВ},$$

де $I_{ЗВ}$ – загальна кількість що подається на ЗВ інформації, причому $I \in I_{ЗВ}$.

Будь-яка інформація, яка не є цільовою інформацією, але що надається на ЗВ або в вирішальну систему в процесі управління, є перешкодою. Наприклад, для ЗВ: $I_E = I_{ЗВ} - I$. У цьому випадку для прийняття рішень цільову інформацію необхідно виявляти (відфільтрувати) на тлі перешкоди.

Контрольні питання

1. Які процеси називаються інформаційними?
2. Які контури управління є в автоматизованій системі?
3. Що є основним джерелом інформації в СУ?
4. З яких частин складається система управління?
5. При виконанні яких умов можна вирішити задачу автоматичного керування?
6. Перерахуйте види інформації.
7. Що таке цільова інформація?

2 РЕАЛЬНІ СИСТЕМИ УПРАВЛІННЯ

2.1 Тренажерні комплекси

В даний час з'являються нові види транспортних засобів і промислових об'єктів, зростає їх складність, підвищуються вимоги до якості управління. Навчання операторів безпосередньо на техніці або об'єкті дорого і не завжди можливо в повному обсязі. З іншого боку робота без підготовки може привести:

- до виходу з ладу різних вузлів і агрегатів і, відповідно, дорогого ремонту;
- до нештатних ситуацій, пов'язаних з ризиком для життя людей.

Потреби в якісному і недорогому навчанні навичкам управління складною технікою визначають необхідність дослідження альтернативних варіантів навчання, без безпосереднього використання техніки.

Такими засобами є різного роду навчальні матеріали: наочні посібники, плакати, відеофільми, що демонструють один або кілька аспектів експлуатації техніки. Ці кошти мають істотний недолік – практично повна відсутність інтерактивності, що вкрай обмежує їх ефективність.

Також, застосовуються спеціальні модифікації реальної техніки, призначені для навчання, в цьому випадку навчання має високу ефективність, але воно дуже дороге і не дозволяє моделювати ситуації пов'язані з небезпекою для людського життя.

У зв'язку з широким розповсюдженням і здешевленням засобів обчислювальної техніки, з'явилися тренажерні комплекси (ТК), які використовують у своїй основі інформаційні системи з комп'ютерним моделюванням в процесі навчання. Такі комплекси позбавлені більшості недоліків традиційних методів навчання, мають невисоку вартість, безпечні для людини і дозволяють моделювати будь-які ситуації, що виникають при експлуатації техніки або промислових об'єктів.

Спектр областей людської діяльності в яких необхідні ТК, надзвичайно широкий, від медицини до управління складними технічними і військовими об'єктами. Основні області застосування ТК:

1) Військова справа. Характеризується високою ціною як обладнання так і "витратних" матеріалів, дорогим техобслуговування; високою складністю освоєння техніки і великим рівнем загрози для життя людини в процесі освоєння. Додатковим фактором є наявність таких ситуацій, які неможливо відтворити іншими шляхами, наприклад ведення бойових дій.

2) Громадянська техніка. Як і військова, найчастіше характеризується високою вартістю і небезпекою при експлуатації непідготовленою людиною. При цьому вартість "витратних" матеріалів і техобслуговування, також може бути висока. Додатковим фактором є те, що цивільні транспортні засоби призначені для перевезення великої кількості людей, що робить ціну помилки екіпажу вкрай високою.

3) Промислові об'єкти. Характеризуються високою вартістю, іноді надзвичайно високою, становлять небезпеку для людини, особливо об'єкти енергетики і хімічної промисловості. Управління великим промисловим об'єктом, як в цілому, так і окремими його частинами представляє масштабну задачу, і для цього потрібна велика кількість підготовленого персоналу, навчання часто тривало за часом і витратна по ресурсам. Додатковим фактором є можливість виникнення позаштатних ситуацій

призводять до техногенних катастроф, які зачіпають велику кількість населення і протяжні площі.

4) Медицина. У цій області важливим фактором є критичність лікарської помилки, яка може привести до смерті людини або серйозних ускладнень.

Таким чином, застосування ТК доцільно в тих областях, де необхідно:

- знизити ймовірність виникнення позаштатних ситуацій;
- зменшити ризик для людського життя;
- поліпшити екологічну обстановку;
- знизити витрати на експлуатацію та навчання;
- моделювати ситуації, які неможливо або вкрай небезпечно відтворити іншим шляхом.

Структурна схема сучасного ТК показана на прикладі тренажера робочого місця механіка-водія танка (рис. 2.1). При побудові структури ТК необхідно враховувати наступні основні положення:

- він повинен бути придатним для індивідуальної підготовки окремого члена екіпажу і підготовки всього екіпажу в цілому;
- він повинен мати таку структуру, яка дозволяла б легко включати цей комплекс до складу групи ТК для проведення тактичної підготовки екіпажів в складі підрозділів;
- повинна бути можливість контролю і оцінки дій операторів.

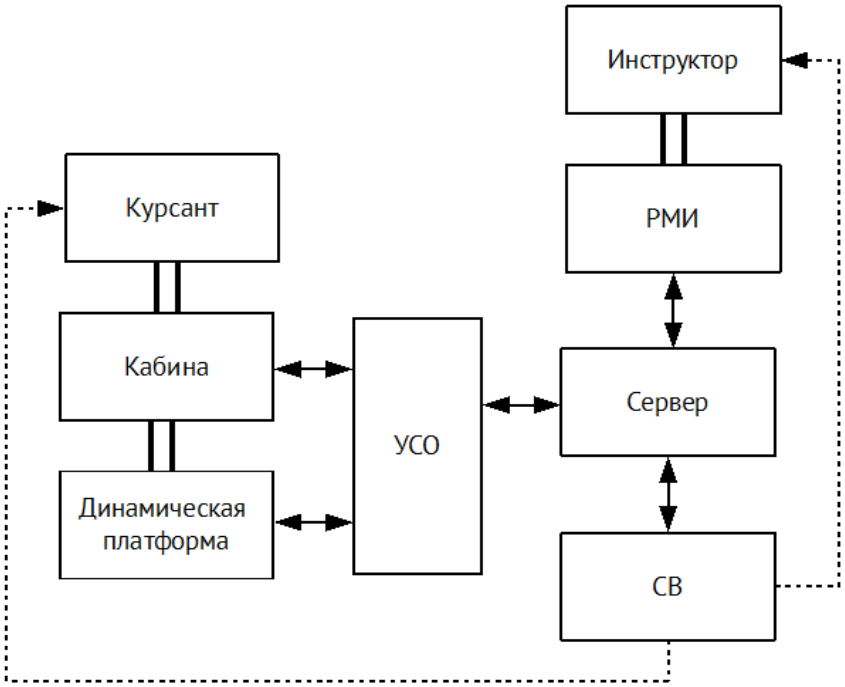

Рисунок 2.1 – Структурна схема ТК

РМІ – робоче місце інструктора; СВ – система візуалізації;

ПСО – пристрій сполучення з обладнанням.

Інструктор за допомогою РМІ задає режим роботи на Сервер (характер вправи, погодні умови, поведінка мішеней і т.д.), який в свою чергу організовує взаємодію

інших частин комплексу. Сервер в своєму складі містить різного роду фізичні і математичні моделі: моделі руху, ландшафту, моделювання балістики, поведінки мішеней, а так само систему обміну даними між підсистемами тренажера.

Керуючі впливи (стан приладів, кути динамічної платформи) на пристрій сполучення (УСО) надходять з Сервера - таким чином, відповідно до встановленого режиму роботи, здійснюється управління динамічної платформою і індикаторами, що знаходяться всередині кабіни.

Назад, на УСО, видається стан органів управління з Кабіни, що в свою чергу дозволяє здійснювати управління моделлю ОУ.

На основі отриманої інформації моделі в складі сервера виробляють нові керуючі впливи, а так само передають в систему візуалізації (СВ) дані необхідні для побудови чергового кадру зображення навколишнього оточення. Візуальна інформація з СВ надходить до Інструкторові і Курсантові – через монітор або проектор.

У даній системі ми можемо виділити два контури переміщення інформації: через Інструктора та Курсанта. При цьому ці два контури є дистанційними, зворотний зв'язок в них здійснюється через СВ.

У ролі ОУ виступає модель технічного засобу, обробка якої відбувається на Сервері. Так само в системі присутній один допоміжний автоматичний контур керування динамічною платформою (на схемі показаний частково).

Таким чином, застосування ТК дозволяє знизити витрати як на процес навчання, так і на вартість самої техніки використовуваної для навчання, при цьому зберігаючи необхідний рівень навчання і його безпеку. За сукупністю характеристик ТК з використанням обчислювальної техніки, є найбільш доцільним засобом для навчання.

Контрольні питання

1. Які переваги дає побудова ТК з використанням засобів обчислювальної техніки?
2. У яких галузях людської діяльності застосовуються ТК?
3. Скільки і які контури управління можна виділити в ТК?
4. Яким чином відбувається рух інформації в ТК?
5. В якому місці (місцях) на структурній схемі ТК знаходиться ОУ?

2.2 Системи управління на електричних станціях і підстанціях

Автоматизована система управління технологічним процесом (АСУТП) - група рішень технічних і програмних засобів, призначених для автоматизації управління технологічним обладнанням на промислових підприємствах. Може мати зв'язок із загальною автоматизованою системою управління підприємством (АСУП). Під АСУТП зазвичай розуміється система, яка забезпечує автоматизацію ключових операцій технологічного процесу на виробництві в цілому або на якомусь його ділянці.

Термін автоматизована система, позначає участь людини в окремих операціях, з метою збереження контролю над процесом або в зв'язку зі складністю або недоцільністю автоматизації окремих операцій. Складовими частинами АСУТП можуть бути окремі системи автоматичного управління (САУ) і автоматизовані пристрої, пов'язані в єдиний комплекс, такі як:

1) Системи диспетчерського управління та збору даних (SCADA) - програмний пакет, призначений для розробки або забезпечення роботи в реальному часі систем збору, обробки, відображення та архівування інформації про об'єкт моніторингу або управління.

2) Розподілені системи управління (РСУ) - найбільш комплексний клас АСУТП. РСУ, як правило, застосовуються для управління безперервними технологічними процесами, до яких можна віднести ті, які повинні проходити днями і ночами, місяцями і навіть роками, при цьому зупинка процесу, навіть короткочасна, неприпустима. Головна вимога до РСУ - відмовостійкість.

3) Інші, дрібніші системи управління, наприклад системи на програмованих логічних контролерах (PLC).

Як правило, АСУТП має єдину систему операторського управління технологічним процесом у вигляді одного або декількох пультів управління, засобів обробки та архівування інформації про хід процесу, типові елементи автоматики: датчики, пристрої управління, виконавчі пристрої. Для інформаційного зв'язку всіх підсистем використовуються промислові мережі.

АСУТП електричних підстанцій є програмно-технічними комплексами - сукупністю засобів обчислювальної техніки, програмного забезпечення і засобів створення і заповнення інформаційної бази при введенні системи в дію і при експлуатації, достатніх для виконання однієї або більше функцій АСУТП.

До складу програмно-технічного комплексу (ПТК) АСУТП підстанцій (ПС) в загальному випадку входять:

- влаштування верхнього рівня (пристрої зв'язку оперативного персоналу з АСУТП і обслуговуючого персоналу з ПТК);
- пристрої рівня приєднань, до яких в даний час відносяться інтелектуальні електронні пристрої (ІЕУ) різного призначення - вимірювальні, управління, контролю стану обладнання, контролю якості, реєстрації аварійних процесів і подій;
- влаштування нижнього рівня, наприклад, що вбудовуються в обладнання контролери, пристрої зв'язку з об'єктом управління (УСО), інтелектуальні датчики і виконавчі механізми ОУ;
- пристрої та лінії зв'язку, що забезпечують обмін інформацією в цифровому вигляді і командами з іншими ПТК і між різними пристроями одного ПТК;

● пристрої цифрового зв'язку з АСДУ верхнього рівня управління і для передачі інформації в АСУ інших суб'єктів;

● пристрої електроживлення ПТК, наприклад, вторинні джерела живлення ПТК і пристрої для підключення зовнішніх силових кабелів електроживлення;

● сервісна апаратура і ЗІП;

● базове і прикладне ПО і документація.

До складу автономних систем (підсистем) автоматичного управління входять підсистеми, що реалізують окремі функції контролю і управління енергетичним обладнанням, а також спеціалізовані мікропроцесорні підсистеми електротехнічного обладнання: релейних захистів і автоматики, протиаварійної автоматики, автоматичного регулювання, система комерційного обліку електроенергії.

Програмно-технічні комплекси АСУТП ПС спільно з іншими технічними засобами повинні забезпечувати:

- ефективне управління процесами перетворення і розподілу електричної енергії;
- єдність системи вимірювань для контролю і управління обладнанням, технічного і комерційного обліку, систем диспетчерського управління;
- спостережуваність параметрів режиму і стану обладнання в нормальних і аварійних режимах;
- управління всіма пристроями, дія яких необхідно для ведення режимів, запобігання відмов обладнання, локалізації та усунення наслідків відмов обладнання зі збереженням живучості підстанції;
- спостереження за станом підстанції, результатом перемикань і діями оперативного персоналу;
- безпеку і надійність роботи автоматизованого обладнання;
- ефективне управління параметрами і економічністю обладнання, яке автоматизується;
- ефективну участь обладнання, яке автоматизується, в управлінні параметрами режиму електричної мережі та енергосистеми в цілому;
- комфортність роботи оперативного та обслуговуючого персоналу, а також оцінку ефективності його дій;
- інформаційний супровід виробничо-технічної діяльності експлуатаційного персоналу.

Таким чином, програмно-технічний комплекс повинен забезпечувати можливість створення однорівневих і багаторівневих, ієрархічних систем розподіленого управління і централізованого контролю, відповідних структурі технологічного об'єкта та характеру управління ним, відкритих для модернізації і розвитку без необхідності зміни раніше реалізованих технічних рішень.

Застосовувані апаратні і програмні засоби мають модульну структуру і розвинуте системне ПЗ. Що допускає широкий діапазон їх використання: від мінімального набору для управління одним приєднанням або виконання однієї функції до максимального, що забезпечує виконання всіх передбачених функцій для всіх рівнів управління ПС.

Обмін інформацією та командами між ПТК, що входять в одну АСУТП може виконуватися з використанням єдиної для системи локальної мережі, по виділеним

цифровим каналам зв'язку або з використанням пристроїв типу "шлюз", що забезпечують контрольований обмін між пристроями, в тому числі, що мають різні інтерфейси. Як пристрої типу "шлюз", що входять до складу ПТК, використовуються:

● спеціалізовані програмно-технічні засоби;

● ПК оснащені відповідним ПО і інтерфейсними картами з необхідним рівнем гальванічного поділу.

Типи каналів зв'язку і протоколи обміну визначаються технологічними вимогами до часу доставки (поновлення) інформації і команд різного виду і призначення, характеристиками і можливостями мікропроцесорних підсистем. Обмін інформацією з автономними підсистемами автоматичного управління є двостороннім.

Також передбачається прийом команд управління і їх оперативне відпрацювання, формування і передача технологічної інформації в АСТУ, включаючи інформацію у вигляді відеопотіку.

АСУТП ПС будується на основі взаємозв'язку функцій автоматизації технологічних процесів основного і допоміжного обладнання, як єдина інтегрована система на основі загальної бази даних (БД).

Базовим поняттям АСУТП і її підсистем є інформаційна модель, яка служить для відображення і опису інформаційних об'єктів, що беруть участь в технологічному процесі реалізації функцій підсистем і АСУТП в цілому (таблиці бази даних, алгоритми, відео, форми, документи). Структура представлення і форма опису інформаційної моделі кожної підсистеми однозначно визначає інформаційні процеси і одержувані інформаційні об'єкти. АСУТП підстанції будується на основі єдиної уніфікованої інформаційної моделі, що забезпечує адекватну взаємодію з верхніми рівнями управління.

В АСУТП підстанцій основні компоненти - функціональні підсистеми, окремі програмно-технічні засоби (пристрої) - об'єднані в єдину систему з використанням методів інтегрування і / або агрегування.

В **інтегрованих системах** всі пристрої пов'язані магістралями певного типу за узгодженими інтерфейсів і протоколів інформаційного обміну.

У **агрегованих системах** пристрої можуть мати різні середовища настройки для своїх частин. Взаємодія між об'єднуються частинами системи здійснюється за стандартним (або спеціальним) інформаційного протоколу обміну, що забезпечує узгодження адресних просторів для даних агрегіруємих підсистем.

Застосування стандартизованих інформаційних моделей і передових систем передачі даних, а також набір відповідних сервісів реального часу забезпечують можливість взаємодії додатків верхнього рівня, дозволяючи використовувати інформаційні моделі, специфічні для даної галузі застосування.

Функціональна структура АСУТП ПС визначається сформованою технологією управління обладнанням ПС. З появою пристроїв ІЕУ, які поряд з функціями збору даних і управління забезпечують функції релейного захисту, реєстрації, контролю технічного стану, архітектура АСУ ТП являє собою розподілену інтелектуальну систему. На рис. 2.2, наведено приклад схеми базової архітектури АСУТП підстанції з ЛВС. З'єднання підстанції з системами і користувачами верхніх рівнів управління являє собою корпоративну ВС. З'єднання від ВС до ЛВС підстанції може виконуватися через маршрутизатор і міжмережевий екран. На схемі так само показаний "виділений канал"

для віддаленого управління, який відображає наявність в системах управління інтерфейсу як з головним підстанції групи (при відсутності постійного персоналу), так і з корпоративної АСУТП. Для зовнішніх підключень може знадобитися пристрій кодування (на схемі не показано).

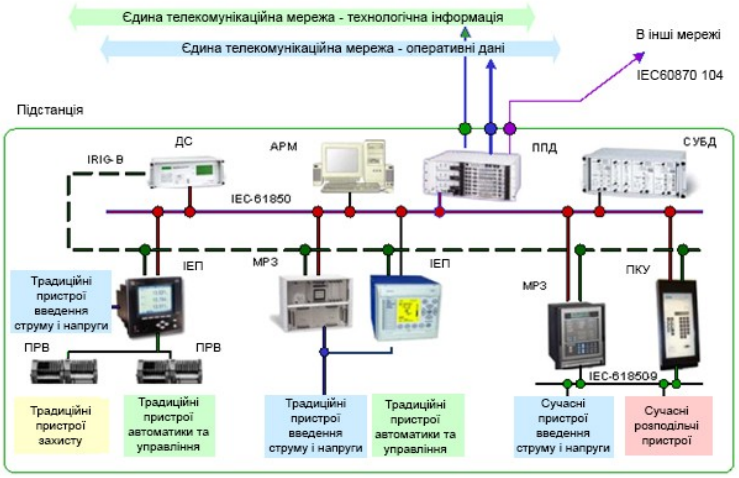

Рисунок 2.2 – Архітектура системи автоматики підстанції

ІЕП – інтелектуальне електронний пристрій; ПРВ – пристрій розподіленого вводу/виводу; МРЗ – мікропроцесорний пристрій РЗ; ПКУ – пристрій контролю і управління; ППД – процесор передачі даних; ДС - джерело синхронізації

На схемі відображені два методи управління і збору даних:

● через пристрої ІЕУ, підключені до обладнання і датчикам підстанції за допомогою спеціалізованої ЛВС.

● шляхом з'єднання мідним дротом (жорсткого з'єднання).

В обох випадках дані і події зберігаються в БД, яка знаходиться на сервері. Якщо потрібна реєстрація подій у часі з точністю ± 1 мс, то точність синхронізації в пристрої ІЕУ повинна бути не гірше ± 0,1 мс.

Показана шина синхронізації, що з'єднує джерело синхронізації (показаний як IRIG-B приймач) з пристроєм ІЕУ, не потрібно, якщо необхідна точність синхронізації забезпечується через мережу ЛВС.

Контрольні питання

1. Які переваги дає побудова АСУТП з використанням засобів обчислювальної техніки?
2. У яких галузях людської діяльності застосовуються АСУТП?
3. Які елементи входять до складу АСУТП ПС?
4. Яким чином відбувається рух інформації в АСУТП ПС?
5. Які вимоги до точності синхронізації при реєстрації подій?

3 АЛГОРИТМИ ІНФОРМАЦІЙНОГО ПОШУКУ

3.1 Оцінка складності алгоритмів

При порівнянні різних алгоритмів, зокрема обробки інформації, важливо знати, як їх швидкодія залежить від обсягу вхідних даних. У загальному випадку швидкодію оцінюють за допомогою **складності алгоритму**. Складність алгоритму можна вирахувати через залежність кількості операцій, що потрібно виконати за алгоритмом, від кількості елементів N, які подаються для обробки. Для позначення складності застосовується так звана **О-нотація**. Якщо при збільшенні кількості вхідних даних ($N \to \infty$), час виконання алгоритму зростає з тією ж швидкістю, що і функція $f(N)$, то кажуть, що такий алгоритм має складність $O(f(N))$.

Якщо алгоритм складається з декількох функціонально незалежних частин, то оцінка його складності виконується відносно тої частини, яка зростає найшвидше. Наприклад, якщо дві частини мають складності $O(N)$ і $O(N^2)$, то оцінка для всього алгоритму буде $O(N^2)$.

Якщо один алгоритм (або його частина) функціонально залежний від іншого алгоритму, то їх складності перемножуються. Наприклад, функція алгоритму А, зі складністю $O(N^2)$, викликається всередині циклу зі складністю $O(N)$, алгоритму Б, тоді для алгоритму Б складність буде $O(N^2) \cdot O(N) = O(N^3)$.

Також, в загальному випадку, для оцінки можна не враховувати константи, які стоять у виразах, $O(2 \cdot N^2)$ еквівалентно $O(N^2)$, за умови $N \to \infty$.

Оцінка складності ведеться для трьох випадків: кращого, середнього і найгіршого. На практиці найбільш цікавими є тільки середній і найгірший (для оцінки найдовшого інтервалу необхідного часу). Вочевидь, оцінка для середнього випадку знаходиться між кращим і гіршим.

Розглянемо оцінку складності на прикладі алгоритму пошуку числа X в довільній чисельній послідовності A[N]. У кращому випадку, якщо елемент X буде першим в послідовності, складність алгоритму буде $O(1)$, тобто не буде залежати від кількості елементів. У гіршому випадку, якщо елемент X розташований найостаннішим, складність буде дорівнювати $O(N)$ — доведеться зробити N кроків, щоб знайти елемент. Ці обидва випадки досить малоймовірні, на практиці елемент X може з'явитися в будь-якому місці послідовності. В цьому випадку оцінка складності буде $O(N/2)$. Це буде усереднений випадок з усіх, які виникають на практиці.

Для низки алгоритмів оцінки для всіх трьох випадків можуть збігатися. Для попереднього прикладу, якщо елемент X взагалі не буде знаходитися в послідовності, то завжди потрібно буде зробити N кроків для визначення цього факту. І оцінка складності буде сягати $O(N)$ у всіх випадках.

Найбільш часто зустрічаються алгоритми з наступними оцінками (розташовані в порядку зростання складності):

$$O(1),\ O(\log N),\ O(N),\ O(N \cdot \log N),\ O(N^2),\ O(N^C),\ O(N!)$$

У таблиці 3.1 показано час виконання алгоритмів з різною оцінкою складності для комп'ютера, який виконує мільйон «типових» операцій в секунду.

Таблиця 3.1 — Час виконання алгоритмів в залежності від виразу оцінки їх складності

N	**10**	**20**	**30**	**40**	**50**
O(N^3)	0,001 секунд	0,008 секунд	0,027 секунд	0,064 секунд	0,125 секунд
O(2^N)	0,001 секунд	1,050 секунд	17,9 хвилин	1,29 днів	35,7 років
O(3^N)	0,059 секунд	58,1 хвилин	6,53 років	386 000 років	$2{,}28\cdot10^{10}$ років
O(N!)	3,630 секунд	77 100 років	$8{,}41\cdot10^{18}$ років	$2{,}59\cdot10^{34}$ років	$9{,}64\cdot10^{50}$ років

У деяких випадках використовується так звана амортизована оцінка. Амортизована оцінка позначається додатковим символом '+'. Наприклад, алгоритм вставки елемента в кінець динамічного масиву оцінюється як O(1)+. Це означає, що спосіб розрахунку оцінки залежить від ряду обставин: якщо в блоці пам'яті, що використовується для зберігання елементів масиву (в більшості випадків), є вільне місце для нового елемента, то, оцінка складності операції вставки в кінець масиву обчислюється як O(1). Але, у тих випадках, коли вільного місця немає, і потрібно перенести блок в ОЗП в якому розташовані елементи масиву в інше місце зі збільшенням його розміру і копіюванням всіх елементів, то оцінка складності операції вставки зміниться на O(N) — що значно більше за оцінку у типових випадках.

Для оцінки алгоритмів також доцільно оцінювати обсяг додатково використаної пам'яті.

3.2 Практичне вимірювання складності алгоритмів

Для вимірювання складності алгоритму доцільно використовувати один з двох його параметрів: тривалість часу роботи і кількість операцій, яку виконує реалізація алгоритму. Недоліком першого варіанту є те, що, час роботи залежить від:

– якості реалізації (коду програми);

– умов компіляції (оптимізації та компілятора);

– умов виконання (процесору, ОЗП, ОС, наявності інших активних процесів).

Це ускладнює порівняння різних алгоритмів один з одним. Тому більш доцільно проводити підрахунок кількості типових (ключових) операцій під час виконання алгоритму.

Для цього необхідно в коді програми додати лічильники, які будуть збільшувати своє значення перед виконанням певних операцій. Лічильник являє собою змінну цілого типу, посилання на яку передається в одному з аргументів функції (реалізації алгоритму), перед початком дослідження значення лічильника встановлюється в 0, а після проведення тестувань, значення зберігається до файлу. Згодом, за збереженими

значеннями можна побудувати графік, який дозволить наочно порівняти практичну оцінку з теоретичною.

Розглянемо процес вимірювання складності роботи алгоритму на прикладі послідовного пошуку у невпорядкованій чисельній послідовності A[N]. Для цього необхідно додати в аргументи функції з кодом алгоритму посилання на лічильник int& cnt, і підчас виконання збільшувати цей лічильник на одиницю перед кожною операцією порівняння, використовуючи оператор інкременту (++cnt).

```
int find_rand_test( int k, const int* A, int N, int& cnt )
{
   for( int i = 0; i < N; ++i )
   {
      ++cnt;
      if( A[i] == k  ) return i;
   } // for( int i = 0; i < N; ++i )
   return -1;
}
```

Перед викликом функції необхідно об’явити змінну cmp_count, яка буде зберігати кількість порівнянь, і ініціалізувати її значенням 0. Передамо посилання на цю змінну до виклику функції:

```
int A[N];
…
int cmp_count = 0;
find_rand_test( k, A, N, cmp_count );
```

Після виклику, змінна cmp_count буде містити кількість операцій порівняння, які були зроблені для пошуку елемента k в послідовності A[N].

3.3 Автоматизація оцінки складності алгоритмів

Дослідження алгоритмів доцільно проводити автоматизованим способом. Для цього необхідно:

– змінювати кількість елементів N в послідовності з певним кроком;

– автоматично створювати задану кількість елементів, на кожному етапі тестування;

– вибирати множину елементів для пошуку (випадково);

– проводити пошук і зберігати результати в файл для подальшої обробки.

Розглянемо оболонку для випробування алгоритмів, яка реалізує перераховані вище вимоги. В заголовковому файлі необхідно визначити максимальну кількість елементів N:

```
const int N = …;
```

У файлі реалізації необхідно створити контейнер для зберігання елементів:

```
int A[N];
```

Відкриємо файловий потік (test.log), в який будуть зберігатися результати випробувань:

```
fstream res( “test.log”,ios_base::trunc | ios_base::out );
```

Ініціалізуємо ГПВЧ значенням таймера, для отримання неповторюваних випадкових послідовностей при кожному запуску програми:

```
srand( time( 0 ) );
```

Далі створимо цикл, який буде задавати розмір послідовності, що тестується, де Ns — кількість зрізів розмірів послідовності для випробувань, minN — мінімальний розмір, а deltaN, приріст розміру на наступному зрізі. Зрозуміло, що minN + deltaN·(Ns-1) <= N. Всередині циклу зі змінною s обчислюється значення кількості елементів N1 для поточного тестування, викликається функція тестування test, до якої передаються як аргументи: файловий потік res, контейнер A та розмір послідовності для тестування N1.

```
for( int s = 0; s < Ns; ++s )
{
   int N1 = minN + s * deltaN;
   test( res, A, N1 );
} // for( int s = 0; s < Ns; ++s )
```

Графічно характер зміни довжини послідовності для запланованих випробувань показаний на рис.3.1.

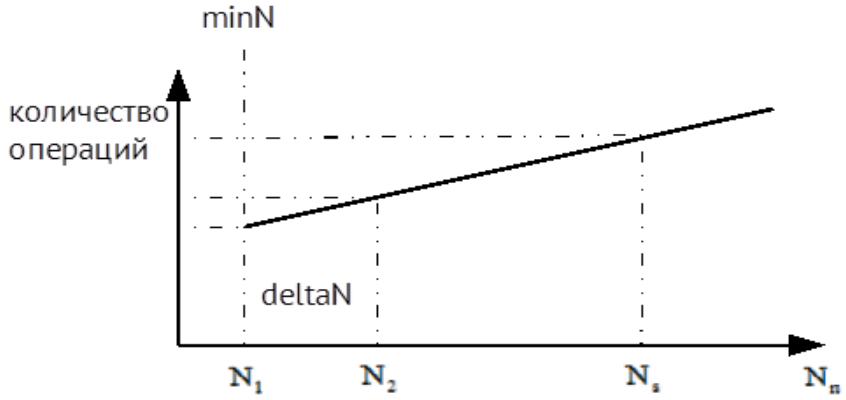

Рисунок 3.1 — Графічне відображення випробувань

Функція тестування test створює послідовність заданого розміру за допомогою функції make_seq, проводить кілька випробувань (задається константою numT) з одним значенням N1, для кожного випробування вибирає елемент для пошуку (функція find_test_key), виконує пошук викликом функції find_rand_test і зберігає усереднені за numT результати тестування в файл res.

```
void test( fstream& res, int* T, int tstN )
{
   make_seq( T, tstN );

   int cmp_count = 0;
   for( int t = 0; t < numT; ++t )
   {
      int kt = find_test_key( T, tstN );
      find_rand_test( kt, T, tstN, cmp_count );
   } // for( int t = 0; t < numT; ++t )

   res << tstN << “ ” << ( cmp_count / numT ) << endl;
}
```

Функція створення послідовності генерує задану кількість tstN випадкових чисел і додає їх до масиву T:

```
void make_seq( int* T, int tstN )
{
   for( int i = 0; i < tstN; ++i )
      T[i] = rand();
}
```

Функція визначення ключа елемента для пошуку find_test_key обирає випадковий елемент з послідовності, для чого використовується генератор випадкових чисел з обмеженням числа розміром контейнера (за допомогою остачі) і повертає його ключ:

```
int find_test_key( const int* T, int tstN )
{
   return T[rand() % tstN];
}
```

Після закінчення тестування файл test.log буде містити кількість виконаних операцій для кожного значення N.

Константи, що визначають параметри тестування: S, numT, minN і deltaN повинні бути об'явлені у заголовковому файлі. Чим більше значення numT, тим більш згладженим (та близькими за характером до теоретичних залежностей) будуть отримані залежності та графіки. Для кожної з функцій доцільно задати первинне визначення, яке складається з назви функції і списку її аргументів і закінчується крапкою з комою.

Якщо тестування проведено коректно, графік отриманої залежності буде повторювати характер теоретичної оцінки складності алгоритму, за досліджуваної операції.

3.4 Інформаційний пошук

Термін **інформація** походить від латинського слова informātiō, що означає роз'яснення, уявлення, визначення, тлумачення. Іншими словами, інформація — це відомості про що-небудь, які сприймаються органами чуття людини.

Види інформації (за ступенем спадання важливості): візуальна, аудіальна, тактильна, нюхова, смакова.

За значущістю інформація буває: актуальна, достовірна, справжня, повна, зрозуміла.

Інформаційний пошук — процес, в ході якого, з загальної послідовності даних вибираються елементи, що відповідають заданим правилам або критеріям. Мета будь-якого пошуку полягає в необхідності знаходити різні види інформації за заданим критерієм, що сприяють отриманню необхідних відомостей і знань для підвищення професійного і культурного рівня, створення нової інформації і формування нових знань, а також для прийняття управлінських рішень. Пошукові засоби і технології, які використовуються для реалізації інформаційних потреб, визначаються тим, яке завдання намагається розв'язати користувач, станом його знань про досліджуваний об'єкт. Крім того, процес взаємодії користувача з системою визначається рівнем знань користувача (повноти подачі, достовірності джерела і т.ін.) та функціональною можливістю системи пошуку. Інформаційні повідомлення, зібрані з різних каналів зв'язку, в подальшому зазнають технічної та наукової обробки.

Технічна обробка інформації полягає в обліку та реєстрації вхідних повідомлень, та перетворення їх до виду зручному для подальшого використання.

Наукова обробка являє собою інформаційний аналіз повідомлень і включає:

1) синтаксичний аналіз — встановлення найважливіших параметрів інформаційного потоку, в тому числі — необхідних кількісних характеристик для вибору технічних засобів подальшої передачі, обробки, зберігання інформації;

2) семантичний аналіз — вивчення інформації з точки зору смислового змісту її окремих елементів;

3) прагматичний аналіз — визначення ступеню корисності інформації, що може бути використана для управління.

Процес, пов'язаний із забезпеченням збереження оброблених даних, відомостей для передачі їх в просторі і часі, називається **зберіганням інформації**.

Повідомлення, реалізовані в певній матеріальній формі, зберігаються у службах:

- документальної інформації (книгосховищах, музеях, архівах і т.д.);
- фактографічної інформації (редакціях газет, телебачення, адресних столах, службах стандартів і т.ін.);
- концептографічної інформації (служби патентної експертизи, прогнозування).

Об'єктом інформаційного пошуку є як сам матеріальний об'єкт, так і його опис.

Залежно від мети, інформаційний пошук можна поділити на адресний і семантичний, від об'єкта — на документальний і фактографічний.

У обчислювальних задачах, пошук це отримання конкретного фрагмента або фрагментів інформації за ключем з великих послідовностей раніше збережених даних. Пошук характерний для багатьох обчислювальних задач. У всіх випадках потрібні ефективні алгоритми пошуку.

Ефективність інформаційного пошуку визначається показниками, що характеризують релевантність знайденої вибірки: повнота і точність видачі інформації. Крім цього, використовуються такі показники, як оперативність, вартість і трудомісткість пошуку.

Мета пошуку у послідовності елементів з ключами — відшукування елементів з ключами, які відповідають заданому критерію пошуку.

Призначення пошуку — отримання доступу до інформації всередині елемента з метою її подальшого використання.

Таблиця пошуку — структура даних у вигляді набору елементів з ключами, яка підтримує дві базових операції: вставку нового елемента і повернення елемента (елементів) за заданим ключем. Таблиці пошуку також називають словниками.

Елемент таблиці пошуку на мові C++ виглядає наступним чином:

```
template <class K, class D> class E
{
   K key;
   D data;

public:

   E() {}
   E( const K& key_, const D& data_ ) : key( key_ ), data( data_ ) {}
   const K& get_key() const { return key; }
   D& access() { return data; }
   bool operator==( const E& e ) const { return key == e.key; }
   bool operator<( const E& e ) const { return key < e.key; }
};
```

Успішний пошук завершується тим, що знайдено хоча б один фрагмент інформації пов'язаний з заданим критерієм пошуку — певним значенням ключа. Результати успішного пошуку називають **влучанням** при пошуку, якщо пошук не дав результатів — **промахом**.

3.5 Послідовний пошук

1) Алгоритм послідовного пошуку в невпорядкованій таблиці. Найпростіший алгоритм пошуку полягає в тому, щоб послідовно переглядати елементи таблиці і порівнювати ключ елемента з ключем пошуку, поки не буде знайдено збіг або не буде досягнуто кінець таблиці. Такий алгоритм називається послідовним або лінійним пошуком.

```
typedef E<K, I> t_elem;
t_elem T[N];
...
int find_rand( const K& key,  const t_elem* T, int N )
{
   for( int i = 0; i < N; ++i )
   {
      if( T[i].get_key() == key )
```

```
      return i;
   } // for( int i = 0; i < N; ++i )
   return -1;
}
```

Для успішного пошуку буде повернено індекс елементу в таблиці, а для неуспішного — число -1 (індекси — невід'ємні цілі числа, тому такого індекса не існує).

Приклад послідовного пошуку у невпорядкованій чисельній послідовності показано на рис.3.2. Вертикальна стрілка показує операцію порівняння.

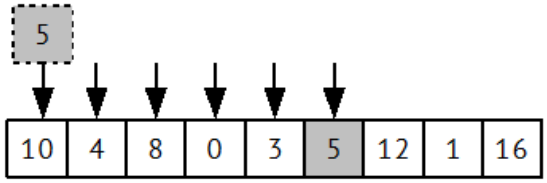

Рисунок 3.2 — Приклад послідовного пошуку числа 5 у невпорядкованій чисельній послідовності

В даному випадку потрібно зробити шість порівнянь для того щоб знайти число 5.

При послідовному пошуку в таблиці символів, що містить N елементів, для виявлення влучання потрібно виконати в середньому близько N/2 порівнянь.

При послідовному пошуку в таблиці символів, що містить N невпорядкованих елементів, завжди необхідно зробити N порівнянь для виявлення промаху.

Але, для виявлення влучень і промахів при послідовному пошуку у впорядкованій таблиці, що містить N елементів, в середньому потрібно виконати близько N/2 операцій порівняння.

Таким чином, якщо частота операцій пошуку значно перевищує таку для операцій вставки — виправдано підтримувати впорядкованість елементів у таблиці. У цьому випадку середня кількість порівнянь при невдалому пошуку скоротиться вдвічі.

2) Алгоритм послідовного пошуку у впорядкованій послідовності. Для упорядкування послідовності необхідно провести її сортування, до виконання пошуку, для чого можна скористатися алгоритмом стандартної бібліотеки:

```
#include <algorithm>
std::sort( T, T + N );
```

Таблиця буде відсортована з використанням оператора '<' (по зростанню). Реалізація алгоритму послідовного пошуку у впорядкованій таблиці наведена нижче:

```
int find_sort( const K& key,  const t_elem* T, int N )
{
   for( int i = 0; i < N; ++i )
   {
      if( T[i].get_key() < key )
         continue;
      if( key < T[i].get_key() )
         break;
```

```
      return i;
   } // for( int i = 0; i < N; ++i )
   return -1;
}
```

В даному алгоритмі для прискорення неуспішного пошуку використовується факт впорядкованості таблиці, і при знаходженні елемента, ключ якого перевищує значення ключа пошуку, алгоритм завершується. В цьому алгоритмі для типу ключа потрібно забезпечити можливість порівняння на меншість (оператор '<').

Приклад послідовного пошуку у впорядкованій чисельній послідовності проілюстровано на рис. 3.3.

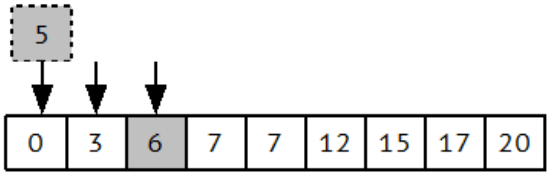

Рисунок 3.3 — Приклад послідовного пошуку числа 5 у впорядкованій чисельній послідовності

В даному випадку потрібно зробити 3 порівняння для того щоб визначити, що числа 5 в послідовності немає — пошук можна зупинити після того, як знайдено число більше за 5. Якщо послідовність буде невпорядкованою, то потрібно зробити 9 порівнянь, щоб перевірити усі числа.

Можна побачити, що при послідовному пошуку будь-якого типу, кількість операцій порівняння лінійно залежить від кількості елементів N, таким чином, при великих значеннях N доцільніше використовувати більш швидкодіючі алгоритми.

3.6 Двійковий пошук

Реалізацію лінійного пошуку в таблицях, в яких елементи розташовані послідовно (в масиві), можна значно прискорити, якщо застосувати підхід «розділяй і володарюй». Для цього, таблицю необхідно розділити на дві частини, визначити до якої з двох частин може належати ключ пошуку, після чого необхідно повторити ті ж дії, над обраною частиною до тих пір, поки в ній є елементи. Такий алгоритм пошуку називається двійковим або бінарним.

Раціональний спосіб реалізації цього алгоритму вимагає підтримки таблиці в упорядкованому вигляді і використання індексів елементів для визначення тієї частини масиву, в якій буде проводитися подальший пошук.

При двійковому пошуку ніколи не виконується більш ніж 2log N порівнянь, як для виявлення влучання, так і для виявлення промаху.

Як і для алгоритму послідовного пошуку у впорядкованої таблиці, перед початком двійкового пошуку, таблицю необхідно впорядкувати.

Для даного алгоритму, як і для попереднього, необхідно забезпечити порівняння на меншість (оператор '<') для типу ключів. На кожному кроці, розмір послідовності елементів що оброблюється, буде скорочуватися в два рази. Пошук закінчиться, або

коли елемент із заданим ключем пошуку буде знайдено, або коли послідовність не буде мати елементів.

Приклад двійкового пошуку у впорядкованої числовій послідовності показано на рис.3.4.

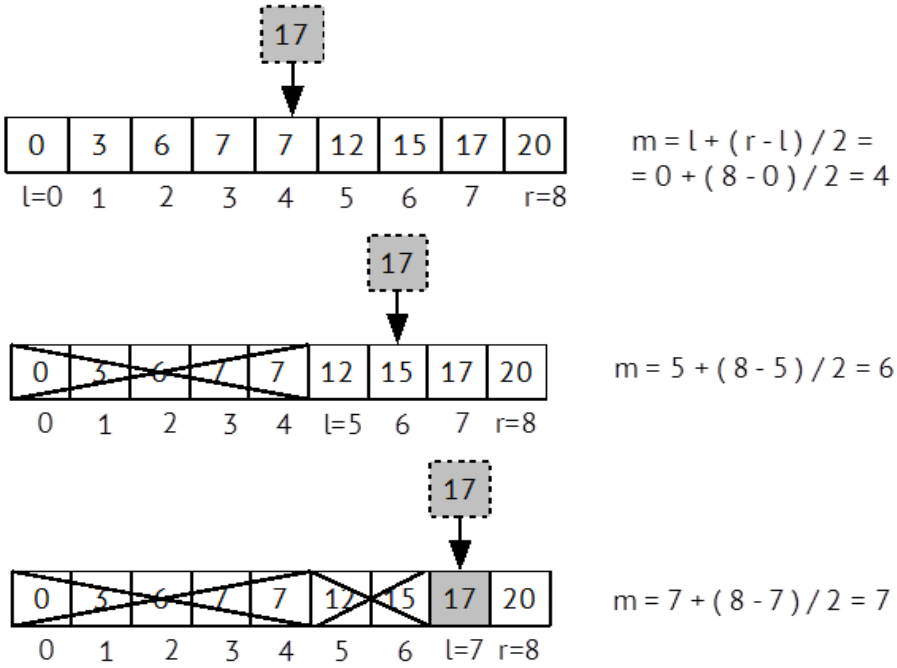

Рисунок 3.4 — Приклад двійкового пошуку числа 17 у впорядкованій чисельній послідовності

В даному випадку потрібно зробити 3 кроки і 6 порівнянь (по два на кожному кроці). Якби використовувався алгоритм послідовного пошуку, знадобилося б 8 порівнянь. Нерекурсивна реалізація алгоритму двійкового пошуку:

```
int bin_search( const K& key,  const t_elem* T, int N )
{
   int l = 0, r = N - 1;
   while( l <= r )
   {    int m = l + ( r - l ) / 2;
      if( key < T[m].get_key() ) r = m - 1;
      else
      if(T[m].get_key() < key ) l = m + 1;
      else
         return m;
   } // while( l <= r )
   return -1;
}
```

Зверніть увагу, що використовується тільки операція порівняння на меншість, для порівняння на більшість аргументи в операції треба поміняти місцями.

Рекурсивний алгоритм двійкового пошуку, може бути реалізовано, шляхом заміни циклу на рекурсивний виклик функції з передачею меж послідовності у аргументах:

```
 int bin_search_rec( const K& key,  const t_elem* T, int l, int r )
 {
   if( l > r ) return -1;
   int m = l + ( r - l ) / 2;
   return key < T[m].get_key() ? bin_search_rec( key, T, l, m – 1 ) :
      T[m].get_key() < key ? bin_search_rec( key, T, m + 1, r ) : m;
}
```

Початковий виклик цієї функції, для пошуку в усій таблиці, виглядає наступним чином:

```
bin_search_rec( key, T, 0, N – 1 );
```

3.7 Інтерполяційний пошук

Вдосконалений алгоритм двійкового пошуку, в якому проводиться більш точне визначення місця розташування ключа в поточному інтервалі пошуку називається інтерполяційний пошук. У цьому алгоритмі на ключі накладається додаткове обмеження — вони повинні бути числами або зводитися до числа. Найчастіше за все, місце поділу послідовності визначається за допомогою лінійної інтерполяції з коефіцієнтом, що вираховується відносно до значень ключів на краях діапазону пошуку. Таким чином, можна «передбачити» місце у таблиці, в якому, швидше за все, знаходиться елемент з ключем пошуку.

Фактично, в порівнянні з алгоритмом двійкового пошуку, в інтерполяційному зміниться тільки процес обчислення місця поділу послідовності m. Якщо для двійкового пошуку формула для обчислення буде така: m = l + a(r - l), де a — константа, яка дорівнює 0.5, то в інтерполяційному пошуку значення a буде обчислюватися на кожному кроці. Замість середнього арифметичного, буде використовуватися співвідношення між тим, як далеко відстоїть ключ пошуку від ключа лівої межі key - lk і різницею ключів правої та лівої меж rk - lk. Тобто a = (key - lk) / (rk - lk), де lk = T[l].get_key( ), rk = T[r].get_key( ).

Приклад інтерполяційного пошуку у впорядкованої чисельній послідовності проілюстровано на рис. 3.5.

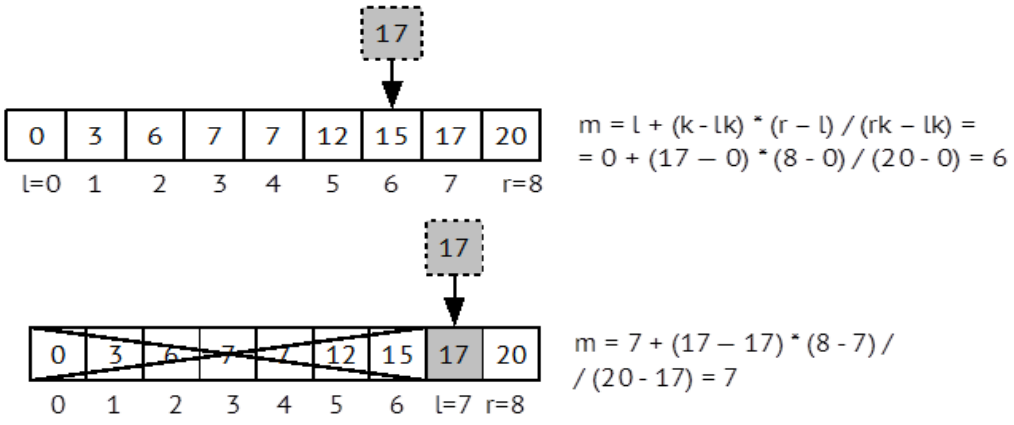

Рисунок 3.5 — Приклад інтерполяційного пошуку числа 17 у впорядкованій чисельній послідовності

В даному випадку потрібно зробити 2 кроки і 4 порівняння (по два на кожному кроці). Реалізація рекурсивного алгоритму інтерполяційного пошуку:

```
int interp_search_rec( const K& key,  const t_elem* T, int l, int r )
{
   if( l > r ) return -1;

   int m = l + (key - T[l].get_key())*(r - l)/(T[r].get_key() - T[l].get_key());

   return key < T[m].get_key() ? interp_search_rec( key, T, l, m – 1 ) :
        T[m].get_key() < key ? interp_search_rec( key, T, m + 1, r ) : m;
}
```

Необхідно пам'ятати, що наведена програма буде коректно працювати тільки для ключів що знаходяться всередині таблиці, якщо ж ключ пошуку менше за мінімальний ключ (крайнього елемента ліворуч) або більше за максимальний (крайнього елемента праворуч), коефіцієнт інтерполяції a буде знаходитись поза діапазоном 0-1, і значення m вийде за межі таблиці. Необхідно або обмежувати коефіцієнт за допомогою перевірок:

```
if( m < l ) m = l;
if( m > r ) m = r;
```

або перед запуском рекурсії перевіряти ключ на входження в діапазон ключів елементів у таблиці. Другий варіант кращий з точки зору швидкодії. Також потрібно перевіряти щоб різниця ключів rk - lk була більша за 0, що зазвичай вимагає ще одного додаткового порівняння на кожному кроці.

Графічно процес інтерполяції показано на рис. 3.6.

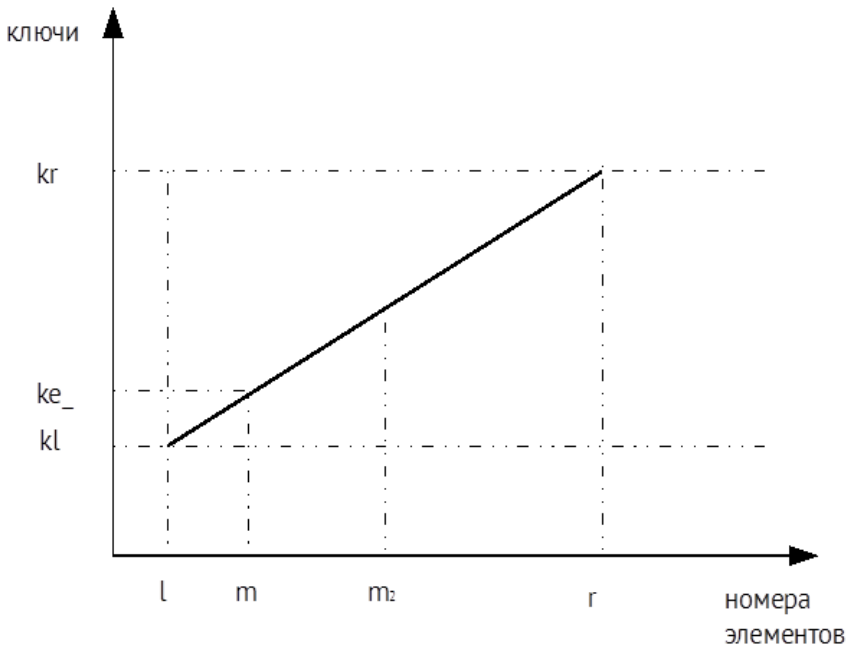

Рисунок 3.6 — Графічне представлення процесу знаходження точки поділу послідовності за допомогою лінійної інтерполяції

Для обчислення співвідношення використовуються операції множення і ділення - ключ повинен бути числовим значенням.

У інтерполяційному пошуку, для кожного влучення або промаху необхідно зробити менше за $2 \log(\log N)$ порівнянь при рівномірному розподілі ключів елементів. На практиці, залежність $\log(\log N)$ можна вважати константою. Якщо $N = 1$ мільярд, то $\log(\log N) < 5$.

В найгіршому випадку, наприклад при зростанні ключів елементів за експонентою, може знадобитися близько N порівнянь. Для великих таблиць, доцільно, на перших кроках пошуку використовувати інтерполяційний пошук, а потім перейти до двійкового, щоб уникнути падіння швидкодії на потенційно «несприятливих» даних з залежністю для ключів за експонентою.

Таким чином, можливо побачити, що за рахунок підвищення вимог до ключів елементів: упорядкованість, представлення у вигляді числа, рівномірний розподіл, можна істотно збільшити швидкодію алгоритму пошуку.

Індивідуальні завдання

1. Побудувати графіки залежності складності від розміру послідовності для алгоритмів лінійного, двійкового та інтерполяційного пошуку.

2. Реалізувати алгоритми лінійного та двійкового пошуку, провести їх дослідження (виміряти кількість операцій порівняння для різної кількості елементів), порівняти результати дослідження з теоретичною оцінкою складності*.

3. Реалізувати алгоритми двійкового та інтерполяційного пошуку, провести їх дослідження (виміряти кількість операцій порівняння для різної кількості елементів, див. додаток А), порівняти результати дослідження з теоретичною оцінкою складності.

4. Написати функцію, яка реалізує нерекурсивний алгоритм інтерполяційного пошуку.

5. Дослідити поведінку алгоритму інтерполяційного пошуку для послідовності з рівномірним розподілом елементів та для послідовності з експоненційним розподілом елементів.

6. Реалізувати як функції гібридний алгоритм пошуку, який у перших трьох кроках пошуку використовує інтерполяційний пошук, а в наступних - двійковий.

7. Дослідити поведінку алгоритмів інтерполяційного та двійкового пошуку на невпорядкованих послідовностях.

8. Побудувати графіки залежності кількості порівнянь від розміру послідовності для алгоритмів лінійного, двійкового та інтерполяційного пошуку.

9. Реалізувати алгоритми лінійного та інтерполяційного пошуку, провести їх дослідження (виміряти кількість операцій порівняння для різної кількості елементів, див. додаток А).

10. Реалізувати алгоритми двійкового та гібридного (інтерполяційного на перших 2-х кроках та двійкового на наступних) пошуку, провести їх дослідження.

11. Дослідити поведінку алгоритмів двійкового та інтерполяційного пошуку для послідовності з рівномірним розподілом елементів та для послідовності з експоненційним розподілом елементів.

1 12. Реалізувати алгоритми лінійного, двійкового та інтерполяційного пошуку, провести їх дослідження (виміряти кількість операцій порівняння для різної кількості елементів), порівняти результати дослідження з теоретичною оцінкою складності.

*Для всіх завдань вважати що дані та ключі збігаються і мають тип int, послідовність для пошуку зберігається з використанням контейнера std::vector. Для отримання елементів послідовності використати генератор випадкових чисел (функція rand). Для впорядкування елементів послідовності використати алгоритм std::sort.

Контрольні питання

1. Що таке інформаційний пошук?
2. Призначення пошуку?
3. З яких частин складається елемент послідовності в якій проводиться пошук?
4. Які вимоги накладаються на ключі елементів для різних алгоритмів пошуку?
5. Які бувають види алгоритмів пошуку?
6. Які переваги дає впорядкування даних перед початком пошуку?
7. Перерахуйте відмінності рекурсивної та нерекурсивної реалізації алгоритму бінарного пошуку.
8. Назвіть особливості інтерполяційного пошуку.
9. Розташуйте алгоритми пошуку в порядку збільшення їх теоретичної швидкодії.
10. Розташуйте алгоритми пошуку в порядку посилення вимог до ключів.
11. Що таке складність алгоритму і як вона визначається?
12. Побудуйте графіки функцій, що найчастіше зустрічаються в якості оцінок складності алгоритмів.
13. В яких межах знаходиться складність алгоритмів пошуку?

4 ХЕШУВАННЯ

Хешування (hashing) — розширений варіант пошуку з індексацією за ключем. При хешуванні застосовується вибірка елементів за ключем з таблиці. Для цього за допомогою арифметичних операцій відбувається перетворення ключа в індекс в таблиці. Така вибірка реалізується через виклик хеш-функції, яка залежить від типу ключа і повертає індекс елемента в таблиці.

Теоретично, при хешуванні пошук відбувається за час, що не залежить від кількості елементів в таблиці.

Основною відмінністю хешування від інших методів пошуку є те, що при хешуванні немає необхідності в операції порівняння на нерівність для ключів (досить побітового порівняння на рівність).

Алгоритми хешування складаються з двох частин:

1) обчислення хеш-функції — перетворення ключа в індекс.

2) вирішення конфліктів в разі збігу індексів у двох ключів двох елементів.

Хешування дозволяє отримати компроміс між часом і обсягом пам'яті, яка використовується для зберігання таблиці. Чим більше пам'яті відводиться під зберігання хеш-таблиці, тим рідше виникають конфлікти і швидше працює пошук.

4.1 Хеш-функції

Хеш-функція — забезпечує перетворення ключа в індекс в хеш-таблиці $h = F_h(K)$. Хеш-функція повинна повертати ціле позитивне число строго менше за H. Для кожного типу ключа в загальному випадку потрібна своя хеш-функція. Приклади найпростіших хеш-функцій для різних типів ключів, для таблиці розміром H:

```
int hash_int( int key ) {   return abs( key ) % H; }
int hash_float( float key ) { return (int) ( fabsf( key ) * ( H-1 ) ) % H; }

int hash_string( const char* key )
{
  int h = 0, a = 127;
  for( ; *key; ++key ) h = ( a * h + *key ) % H;
  return h < 0 ? ( h + H ) : h;
}
```

Ідеальна хеш-функція — така функція, для якої ймовірність конфлікту між індексами для двох різних ключів в таблиці розміром H, дорівнює в точності 1/H. Така функція недосяжна на практиці, але можливо зробити функцію, яка б була близька до ідеальної.

Вимоги до "ідеальної" хеш-функції:

- для обчислення індексу необхідно використовувати всі розряди ключа;
- рівномірність розподілу індексів в таблиці при випадковому виборі ключів.

```
int hash_string_near_ideal( const char* key )
{
   int h = 0, a = 31415, b = 27183;
   for( ; *key != 0; ++key, a = a * b % ( H – 1 ) )
   {
      h = ( a * h + *key ) % H;
   }
   return h < 0 ? ( h + H ) : h;
}
```

В основі цієї хеш-функції лежить **лінійний конгруентний генератор псевдовипадкових чисел** (ГПВЧ):

$$x_i = (a \cdot x_{i-1} + c) \bmod m,$$

де x_i, x_{i-1} – псевдовипадкові числа, a, c та m − параметри генератора.

Вочевидь, що такий генератор завжди повертає один і той же x_i для відповідного x_{i-1}, таким чином, операції вставки і пошуку будуть працювати з одним і тим же індексом, який поверне хеш-функція заснована на такому ГПВЧ. Доцільно використовувати ГПВЧ з параметром $c = 0$, наприклад, наведений в коді вище або ГПВЧ Льюїса, Гудмана, Міллера з параметрами: $a = 7^5$, $c = 0$, $m = 2^{31} - 1$.

4.2 Вирішення конфліктів при хешуванні

У тому випадку, коли хеш-функція для двох ключів дає однаковий індекс в хеш-таблиці, виникає **конфлікт**. Для любої хеш-функції рано чи пізно такий конфлікт виникне, оскільки хеш-таблиця має кінцевий розмір. Таким чином, для функціонування хеш-таблиці на практиці обов'язково потрібен алгоритм вирішення конфлікту.

4.2.1 Роздільне зв'язування — метод вирішення конфліктів, заснований на тому, що в кожній комірці хеш-таблиці зберігається не один елемент, а список елементів. При збігу ключів проводиться вставка нового елемента в кінець списку. А при пошуку проводиться лінійний пошук у списку з індексом, який видає хеш-функція.

Приклад додавання елемента в хеш-таблицю з роздільним зв'язуванням показаний на рис. 4.1. Номерами показані індекси списків, а пунктиром елементи в кожному списку.

Таким чином, роздільне зв'язування зменшує кількість порівнянь що потрібно зробити при пошуку в H раз (в середньому), при використанні списків для H комірок.

У хеш-таблиці, яка містить H списків і N ключів, при використанні роздільного зв'язування ймовірність того, що кількість ключів в кожному списку незначно відрізняється від N/H, близька до 1 при застосуванні хеш-функції близької до ідеальної.

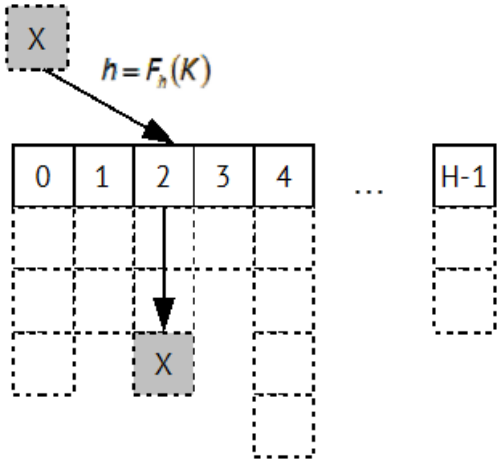

Рисунок 4.1 – Додавання елемента X в хеш-таблицю з роздільним зв'язуванням

4.2.2 Лінійне зондування — метод зберігання ключів в таблиці розміром H > N, при якому розв'язання конфліктів ґрунтується на наявності порожніх комірок в хеш-таблиці. При виникненні конфлікту перевіряється кожен наступний елемент в таблиці до тих пір, поки не буде знайдено порожню комірку. При пошуку процес відбувається аналогічно до тих пір, поки не буде знайдений елемент із заданим ключем пошуку або порожня комірка (це означає що елементу з ключем пошуку немає).

Для лінійного зондування вводиться коефіцієнт завантаження таблиці $\alpha = N/H$. Вочевидь, що для підвищення швидкодії необхідно прагнути до зменшення α, що може досягатися, наприклад, збільшенням розміру таблиці. У міру того як $\alpha \to 1$ в таблиці з'являються безперервні групи зайнятих комірок, які називаються **кластерами**. Наявність протяжних кластерів веде до зниження швидкодії операцій пошуку та вставки.

Приклад додавання елемента в хеш-таблицю з лінійним зондуванням наведено на рис. 4.2. Сірим кольором показані заповнені елементи, комірки 2-4 утворюють кластер, і для знаходження місця вставки потрібно 3 зондування (відзначені горизонтальними стрілками). Елемент X буде вставлений в клітинку з індексом 5.

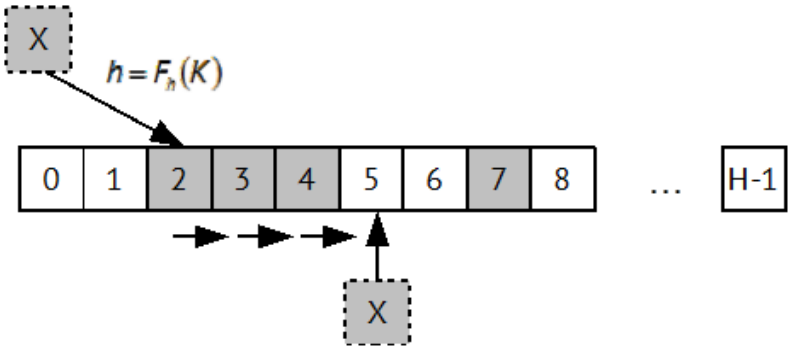

Рисунок 4.2 — Додавання елемента X в хеш-таблицю з лінійним зондуванням

При вирішенні конфліктів за допомогою лінійного зондування середня кількість операцій, необхідних для пошуку в хеш-таблиці розміром H, яка містить $N = \alpha \cdot H$ ключів, дорівнює $ch_{лз} = 0{,}5 \cdot (1+1/[1-\alpha])$ і $cm_{лз} = 0{,}5 \cdot (1+1/[1-\alpha]^2)$ для влучень і промахів відповідно.

4.2.3 Подвійне хешування — розвиток методу лінійного зондування, який дозволяє уникнути створення протяжних кластерів (ефекту кластеризації). При виникненні конфлікту використовується друга хеш-функція $s = F_s(K)$ для отримання кроку зондування s. Наприклад, для ключа цілого типу:

```
int hash_step( int key ) { return ( abs( key ) % 97 ) + 1; }
```

Така хеш-функція не повинна повертати 0. Розмір хеш-таблиці повинен бути взаємно простим числом з кроком, що повертає друга хеш-функція, в іншому разі, не всі комірки таблиці будуть використані для будь-якого кроку.

Якщо друга хеш-функція завжди повертає 1, то подвійне хешування буде повністю еквівалентно лінійному зондуванню.

Приклад додавання елемента в хеш-таблицю з подвійним хешуванням з кроком 4 наведено на рис. 4.3. Для знаходження місця вставки потрібно одне зондування. Елемент буде вставлений в клітинку з індексом 6.

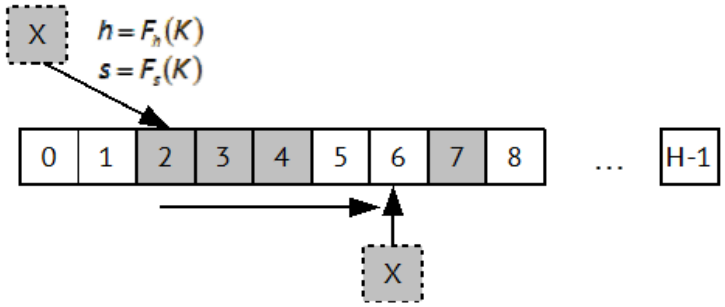

Рисунок 4.3 — Додавання елемента X в хеш-таблицю
з подвійним хешуванням (h = 2, s = 4)

При вирішенні конфліктів за допомогою подвійного хешування, середня кількість операцій, необхідних для пошуку в хеш-таблиці розміром H, яка містить $N=\alpha \cdot H$ ключів, дорівнює $ch_{пх} = (1/\alpha)\cdot\ln(1/[1-\alpha])$ і $cm_{пх} = 1/(1-\alpha)$ для влучень і промахів відповідно. В таблиці 4.1 наведені кількості операцій для двох методів розв'язування конфліктів.

Таблиця 4.1 — Приблизна кількість операцій для лінійного зондування та подвійного хешування в залежності від завантаження таблиці

Завантаження таблиці, α	Влучання		Промах	
	Лінійне зондування	Подвійне хешування	Лінійне зондування	Подвійне хешування
0,1	1	1	1	1
0,5	2	1	3	2
0,75	3	2	9	4
0,9	6	3	50	10
0,95	10	3	200	20

Як видно зі співвідношень та таблиці 4.1, алгоритм подвійного хешування вимагає меншої кількості операцій, ніж лінійне зондування при однакових розмірах хеш-таблиці і її наповненості.

Таким чином, зберігаючи значення α менше за $1-1/\sqrt{t}$ для лінійного зондування і $1-1/t$ для подвійного хешування, можна забезпечити, щоб у середньому для виконання пошуку в таблиці було потрібно менше ніж t операцій. Для хеш-таблиць не можна гарантувати час виконання всіх операцій, але можна гарантувати, що середні витрати на одну операцію будуть невеликими і складність пошуку в хеш-таблиці можна оцінити як O(1)+.

4.2.4 Динамічні хеш-таблиці. Для лінійного зондування і подвійного хешування потрібно зберігати розмір таблиці гарантовано більшою за кількість елементів і підтримувати необхідне значення α, що, при заповненні таблиці, неминуче вимагатиме збільшення її розмірів. Доцільно, кожен раз, коли α перевищить деякий поріг, що визначається вимогою до швидкодії і пам'яті що використовується, збільшувати таблицю, наприклад, в 2 рази. При зміні розміру таблиці потребується її повна перебудова.

При видаленні елемента з хеш-таблиці, що використовує лінійне зондування, необхідно повторно вставити всі елементи кластера, які розташовані за елементом що видаляється. А для подвійного хешування треба повністю перебудувати таблицю.

Індивідуальні завдання

1. Розрахувати кількість операцій ch_{1z} та cm_{1z} для лінійного зондування, для значень a={0, 0.25, 0.5, 0.75, 0.95}.

2. Розрахувати кількість операцій ch_{2h} та cm_{2h} для подвійного хешування, для значень a={0, 0.25, 0.5, 0.75, 0.95}.

3. Розрахувати кількість операцій ch та cm для лінійного зондування та подвійного хешування, для a={0, 0.25, 0.5, 0.75, 0.95} та провести порівняння двох цих методів.

4. Написати функцію операцію вставки елемента в хеш-таблицю з використанням роздільного зв'язування*.

5. Написати функцію, яка здійснює пошук елемента в хеш-таблиці з використанням роздільного зв'язування.

6. Реалізувати у вигляді окремих функцій операції вставки та пошуку у хеш-таблиці з лінійним зондуванням.

7. Реалізувати у вигляді окремих функцій операції вставки та пошуку у хеш-таблиці з подвійним хешуванням.

8. Написати функцію, яка коректно видаляє елемент з хеш-таблиці (таблиця повинна залишатися функціональною) при використанні лінійного зондування.

9. Написати функцію, яка коректно збільшуватиме розмір хеш-таблиці в 2 рази при використанні лінійного зондування.

10. Реалізувати функції, які виконуватимуть операції вставки, пошуку та видалення елемента в хеш-таблиці використовує роздільне зв'язування.

11. Реалізувати функції, які виконуватимуть операції вставки, пошуку та видалення елемента в хеш-таблиці, що використовує лінійне зондування.

12. Реалізувати функції, які виконуватимуть операції вставки, пошуку та видалення елемента в хеш-таблиці, що використовує подвійне хешування.

*Для всіх завдань вважати що, дані та ключі збігаються і мають тип int, таблиця зберігається з використанням контейнера std::vector. Для представлення списків використовувати std::list.

Контрольні питання

1. Що таке хешування?

2. Назвіть відмінності пошуку по хеш-таблиці від пошуку у впорядкованій послідовності.

3. Що таке хеш-функція?

4. Яким вимогам повинна відповідати хеш-функція?

5. Що таке конфлікт при пошуку в хеш-таблиці?

6. Які існують методи розв'язання конфлікту в хеш-таблиці?

7. Що таке кластеризація в хеш-таблиці?

8. Для чого потрібен коефіцієнт завантаження хеш-таблиці?

9. Як визначити розмір хеш-таблиці, якщо відоме обмеження на середню кількість операцій і кількість елементів в таблиці?

10. Побудуйте графіки залежності необхідної кількості операцій для пошуку від коефіцієнта завантаження таблиці, для методів лінійного зондування і подвійного хешування.

Список рекомендованої літератури

1. Бусяк Ю.М. Побудова структур даних обміну інформацією між підсистемами тренажерів транспортних засобів/ Ю. М. Бусяк, О. Г. Васильченков - Вісник Національного технічного університету "Харківський політехнічний інститут". Збірник наукових праць. Тематичний випуск: Автоматика та приладобудування. - Харків: НТУ "ХПІ". – 2002. - №9, т. 7. – 198 с.
2. Фещенко А. Потенціал та перспективи вітчизняного військового тренажеробудування / А. Фещенко - Defence Express №4, Апр. 2006. - С. 39-45.
3. Рекомендації щодо застосування основних структурних схем та вимоги до організації АСУТП підстанцій 110 - 750 кВ з урахуванням функціональної достатності та надійності, 2010 р.
4. Stroustrup, Bjarne. The C++ programming language / Bjarne Stroustrup.— Fourth edition. Published by Pearson Education, Inc. 2013. 1360 p. ISBN 978-0-321-56384-2.
5. Грицюк Ю., Рак Т. Програмування мовою С++. Навчальний посібник. / Юрій Грицюк, Тарас Рак. Львів. Вид-во ЛДУ БЖД 2011. 290 с. ISBN 978-966-3466-85-9.
6. Мелешко Є.В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. – Кропивницький: Видавець – Лисенко В.Ф., 2019. – 156 с.
7. Sedgewick R. and Wayne K. Algorithms 4th Edition. Robert Sedgewick and Kevin Wayne. Pearson Education, Inc. 2011. 955 p. ISBN-13: 978-0-321-57351-3.
8. Алгоритми і структури даних: лабораторний практикум для студентів напряму підготовки "Телекомунікації і радіотехніка" / Уклад.: М.А. Скулиш, С.В. Суліма,. – К.: КПІ ім. Ігоря Сікорського, 2021. –109с.
9. Michael T. Goodrich, Roberto Tamassia. Section 6.3.3: Linear Probing // Algorithm Design and Applications. — Wiley, 2015. — С. 200–203. — ISBN 978-1-118-33591-8.
10. Pagh, Rasmus; Rodler, Flemming Friche (2001). "Cuckoo Hashing". Algorithms — ESA 2001. Lecture Notes in Computer Science. Vol. 2161. CiteSeerX 10.1.1.25.4189. doi:10.1007/3-540-44676-1_10. ISBN 978-3-540-42493-2.
11. Kirsch, Adam; Mitzenmacher, Michael D.; Wieder, Udi (2010), "More robust hashing: cuckoo hashing with a stash", SIAM J. Comput., 39 (4): 1543–1561, doi:10.1137/080728743, MR 2580539.
12. P. Celis. Robin Hood Hashing. PhD thesis, Computer Science Department, University of Waterloo, April 1986. Technical Report CS-86-14.

ЗМІСТ

Навчальне видання

ЗУЄВ Андрій Олександрович
ІВАШКО Андрій Володимирович
ГАПОН Дмитро Анатолійович

ІНФОРМАЦІЙНИЙ ПОШУК

Методичні вказівки
до виконання самостійних робіт

з курсу «Інформаційні технології та програмування»

для студентів спеціальностей «Автоматизація та комп'ютерно-інтегровані технології», «Телекомунікація та радіотехніка» усіх форм навчання вищих навчальних закладів

Відповідальний за випуск Зуєв А.О.
Роботу до друку рекомендував Дудник О.В.

План 2022 р., Поз.343
Підписано до друку 05.11.2022. формат 60×84 1/16. Папір друк. № 2.
Друк – різографія. гарнітура Times New Roman. Розум. друк. арк.3,2.
Обл. – вид. арк. 2,7.Наклад 100 прим. Зам. № . Ціна договірна.

Самостійне видання