МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

МІЖНАРОДНИЙ КЛАСИЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ ПИЛИПА ОРЛИКА

Економічно-технологічний факультет

Кафедра інженерних технологій

КВАЛІФІКАЦІЙНА РОБОТА

другого (магістерського рівня) вищої освіти

на тему:

«Аналіз методів зменшення обсягу маршрутних даних для систем зберігання інформації про рухомий об'єкт моніторингу»

Зі спеціальності 123

«КОМП’ЮТЕРНА ІНЖЕНЕРІЯ»

Виконавець:

Бортник М.А.

Науковий керівник:

к.т.н., доц. Гайша О.О.

Миколаїв – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ASP	Active Server Pages
CSS	Cascading Style Sheet
CSV	Comma Separated Values
GUI	Graphics User Interface
HTML	HyperText Markup Language
JSP	Java Server Pages
MTBF	Mean Time Before Failure
PC	Personal Computer
PHP	Personal Home Page, **P**HP **H**ypertext **P**reprocessor
SQL	Structured Queries Language
WWW	World Wide Web
XML	eXtensible Markup Language
БД	База даних
ЗІ	Захист інформації
ІТ	Інформаційні технології
ОО	Об’єктно-орієнтований (-на, -не)
ООП	Об’єктно-орієнтоване програмування
ПЗ	Програмне забезпечення
ПК	Персональний комп’ютер
СВС	Система відео спостереження
СКПМ	Система контролю поточного місцезнаходження
СУБД	Система управління базами даних

ВСТУП

Розвиток комп’ютерної техніки та супутніх інформаційно-комунікаційних технологій вніс численні зміни у багато сфер життя як окремої людини, так і суспільства в цілому. Зокрема, значно змінилися засоби та відповідні підходи до забезпечення безпеки самих різноманітних систем та сфер нашого життя. Чого варте лише повсюдне використання систем відеоспостереження (далі – СВС), які на сьогоднішній день стали одним із основних засобів розкриття злочинів тяжкого характеру та запобігання легких. Однак, зростаюча кількість камер робить їх роботу все більш автоматичною: усю інформацію вони пишуть в архіви, а поточну картинку, зазвичай, ніхто і не переглядає, і, тільки при виникненні такої необхідності, потрібні ділянки відео витребують з цих мовчазних цифрових архівів. Відповідно, виникає проблема збереження величезних обсягів цифрових даних, що продукуються десятками, сотнями і тисячами камер у місті, десятками тисяч камер у країні і т.д. З абсолютно аналогічною проблемою стикаються і інші системи безпеки, такі, як наприклад, системи контролю поточного місцезнаходження (далі – СКПМ) рухомих об’єктів.

Ці системи, як і СВС, мають певні налаштування, які можуть сприяти зменшенню обсягів даних, що фіксуються, але це стає можливим тільки за рахунок зменшення «якості» відомостей, що зберігаються для даної предметної галузі. Деякі «погіршення» інформації є цілком прийнятними, деякі навпаки можуть змінювати зміст відомостей, що містяться у такій інформації, на десятки відсотків, чи більше (що є цілком неприпустимим). Відповідно, важливою і **актуальною** задачею є вироблення методів для одночасно «безпечної» і ефективної мінімізації даних про маршрути рухів контрольованого об’єкту (об’єктів).

Таким чином, **метою** роботи можна вважати ефективне зменшення обсягів маршрутних даних, що мають зберігатися у відповідному архіві для забезпечення усебічної безпеки процесу переміщення рухомих об’єктів.

Для досягнення поставленої мети необхідно вирішити наступні **задачі** дослідження:

- проаналізувати існуючі способи зняття, перетворення та зберігання маршрутної інформації з оглядом на використання можливостей зменшення її обсягів;

- розробити алгоритм мінімізації маршрутних даних на основі відомих алгоритмів, виконуючи їх оптимізацію, а також необхідні супутні інженерні рішення (як, наприклад, формат збереження даних та ін.);

- здійснити програмну реалізацію розробленого алгоритму та її комплексне дослідження;

- використовувати особливості процесу мінімізації з урахуванням різноманітних залежностей (зокрема, від швидкості руху об’єкта), вимог надійності збереження, захисту відповідної маршрутної інформації.

Об’єктом дослідження є процес накопичення та мінімізації маршрутних даних рухомих об’єктів.

Предметом дослідження є методи та моделі мінімізації маршрутних даних.

В роботі використані загальнонаукові **методи** аналізу (для огляду існуючих рішень та принципів їх роботи) та синтезу (для створення власних рішень). Для опису кривих маршрутної інформації застосовано методи вищої математики, для розробки програмної реалізації та усіх супутніх елементів – методи галузі інформаційних технологій.

В роботі отримано нові науково-технічні результати:

- розроблена процедура моделювання для отримання маршрутної інформації;

- проаналізовано можливості застосування методів вищої математики (сплайн-інтерполяції, інтегральних перетворень) для стиснення саме маршрутної інформації сільськогосподарської техніки (з урахуванням специфіки такої інформації);

- на основі наукового аналізу запропоновано новий комплексний алгоритм стиснення маршрутної інформації;

- створено власний формат збереження маршрутної інформації.

Робота реалізована до робочого програмного продукту, тому може мати реальне практичне значення, а саме, застосовуватися в системах мінімізації маршрутної інформації у централізованих файлових архівах. В перспективі може бути розширена для урахування специфіки не лише сільськогосподарської, а й комунальної спецтехніки, маршрутного транспорту, і т.п.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО ВИРІШЕННЯ ПРОБЛЕМИ ЗБЕРІГАННЯ МАРШРУТНОЇ ІНФОРМАЦІЇ ДЛЯ СИСТЕМ ЦЕНТРАЛІЗОВАНОГО МОНІТОРИНГУ

1.1. Області застосування маршрутної інформації.

Вище в загальних рисах була окреслена доцільність процесу збереження маршрутної інформації та її широке, різнопланове використання в сучасних системах захисту, контролю та управління. Розглянемо докладніше, де конкретно та з якими цілями може застосовуватися вказана маршрутна інформація.

По-перше, і найголовніше, системи контролю поточного місцезнаходження (СКПМ) застосовуються на спеціалізованій рухомій техніці за умови, що власник техніки та її оператор (водій) є різними особами. За такої умови цілком природним є бажання власника (або його представника, або іншої уповноваженої особи) мати можливість здійснювати перевірку використання його технічних засобів строго у відповідності до їх призначення. В першу чергу, це стосується місцезнаходження рухомого складу техніки, наприклад, як у наступних трьох випадках.

а) поточне розташування автобусів комунальних підприємств може перевірятися будь-якою людиною, як членом громади, якій належить техніка; це на сьогоднішній день реалізується у вигляді виділеної веб-сторінки, куди може вільно заходити будь-хто з користувачів мережі Інтернет, та передивлятися місцезнаходження транспорту, яке відслідковується по GPS-трекерам, або по датчикам системи ГЛОНАСС;

б) також часто під час зимової непогоди обивателів цікавить місцезнаходження спеціальної снігозбиральної техніки, на яку, як і на звичайний транспорт, навішують GPS-модулі, і будь-хто на сайті Служби автодоріг (або аналогічної) може подивитися положення усіх її транспортних засобів, оцінити ефективність їх використання, поскаржитися, або подякувати керівництву організації;

в) надзвичайно актуальною для України, як для держави, що має надпотужний аграрний потенціал, є задача відстеження положень сільськогосподарської техніки її власником (або, найчастіше на практиці, представниками власника – уповноваженими особами). Дійсно, нажаль в реальності нерідкою є ситуація, при якій водії транспортних засобів не виконують своєї роботи, паливо зливають та продають, заявляючи, що робота виконана і паливо витрачене за призначенням. При цьому поле залишається не политим, або не переораним, і т.п. Звичайно, ця ситуація вкрай негативно впливає на продуктивність всього процесу сільськогосподарського виробництва і, отже, є неприпустимою.

Для коректності подальшого викладення матеріалу будемо називати ситуацію, коли рухома техніка з волі оператора (водія) не рухається по заданому маршруту, а використовується (чи не використовується) іншим чином, словами «несанкціоноване використання рухомої техніки» (НВРТ).

Розглянемо останній випадок докладніше, адже він має суттєві особливості, по відношенню до інших описаних вище ситуацій: тут маршрутна інформація не є потрібною негайно, а може переглядатися відстрочено, через кілька годин, наступного дня, чи, навіть тижня. В той же час, наприклад, поточна інформація про наявне положення громадського транспорту потрібна пасажирам безпосередньо у даний момент часу, при очікуванні на зупинці, а після здійснення поїздки, чи навіть після посадки в автобус, моментально втрачає свою актуальність. Аналогічно і інформація про положення снігоприбиральної техніки є важливою в основному для поточного моменту часу, адже при плануванні маршруту, якими вулицями повертатися додому, або якщо людина застрягла у заносі, важливо де рятувальна техніка працює в даний момент часу, але після виїзду особи із засніжених ділянок ця інформація різко втрачає для неї свою цінність.

Отже, маршрутна інформація саме про переміщення сільськогосподарської техніки відрізняється тим, що має зберігатися певний (не малий, не менше величин порядку тижня) інтервал часу, а, отже, потребує

відповідних накопичувачів інформації (архіву), що, звичайно, коштують певних грошей. Очевидно, збільшення інтервалу часу ΔT, протягом якого зберігаються маршрутні дані, підвищує вартість системи і зменшує її загальну економічну ефективність. З іншого боку, бездумне зменшення цього параметру може сприяти підвищенню імовірності виникнення непоміченого НВРТ, що буде нести для компанії збитки, описані вище.

Таким чином, очевидною для цього випадку є наявність конфліктної ситуації: збільшення часу зберігання приводить до збільшення вартості цифрових носіїв для ведення архіву, але зменшення цього часу ΔT приводить до збільшення втрат на НВРТ. Очевидно, що вибір найкращого значення ΔT являє собою один із різновидів математичної задачі оптимізації, яка, тим не менше, має стохастичну складову, оскільки важко передбачити НВРТ для будь-якого водія (імовірність навіть може відрізнятися для одного і того самого водія, залежно від його поточних життєвих обставин, тому звичайно, краще мінімізувати цю можливість та регулярно повідомляти/нагадувати водіям про безперервний моніторинг поточного місцезнаходження їх одиниці рухомої техніки). Обґрунтування вибору конкретних значень ΔT буде виконуватися нижче у відповідному розділі даної роботи.

Якщо місцевість має складний рельєф, то використання маршрутної інформації може також носити і довідковий або орієнтовний характер: один раз прокладений зручний та безпечний маршрут для виконання певних сільськогосподарських робіт (наприклад, поливу такої нерівної ділянки) може повторюватися іншими водіями в інші періоди на основі збереженої маршрутної інформації «першопрохідця», що міг ще і затратити певний час на пошук найкращого шляху в умовах пересіченої місцевості. Хоча в цілому, цей варіант застосування маршрутних відомостей є набагато більш екзотичним (хоча і можливим на практиці), а головним способом використання залишається все ж таки контроль адекватності, санкціонованості використання рухомої сільськогосподарської техніки

відповідно до загальних потреб її власника, а не ситуативних бажань оператора (водія).

1.2. Способи зняття маршрутної інформації та оцінка її обсягів.

Таким чином, у попередньому підрозділі встановлено, що маршрутна інформація має широке застосування з метою контролю, тобто для забезпечення різнопланової безпеки рухомої техніки та всієї системи, у яку вона включена, в цілому. Наступним кроком має бути оцінка обсягів такої інформації (причому з вибором конкретних оптимальних значень ΔT), але для цього слід спочатку розібратися зі способом зняття маршрутної інформації, тобто зрозуміти її суть.

В першу чергу, при розгляді будь-яких об’єктів на поверхні землі слід згадати про те, що наша планета має наближено форму кулі (хоча і дещо стиснутої, але для наших цілей це абсолютно не є принциповим), а отже, будь-яка точка на її поверхні може характеризуватися двома числами: широтою та довготою. Ці величини є кутами, що:

- утворює радіус-вектор точки на поверхні Землі із площиною екватора – географічна широта;

- утворює діаметральна площина, яка проходить через дану точку та вісь обертання Землі, з одного боку, та спеціальною меридіональною площиною, що також проходить через вісь обертання Землі та точку, довгота якої прийнята рівною нулю (Грінвічський меридіан) – географічна довгота.

Точність визначення широти і довготи у сучасних системах навігації є досить високою та дозволяє розрізняти точки, що лежать на поверхні Землі одна від одної на відстанях порядку метра.

Типовий приклад завдання координат точки місцевості, що надається системою Google Maps наведено на рис. 1.1.

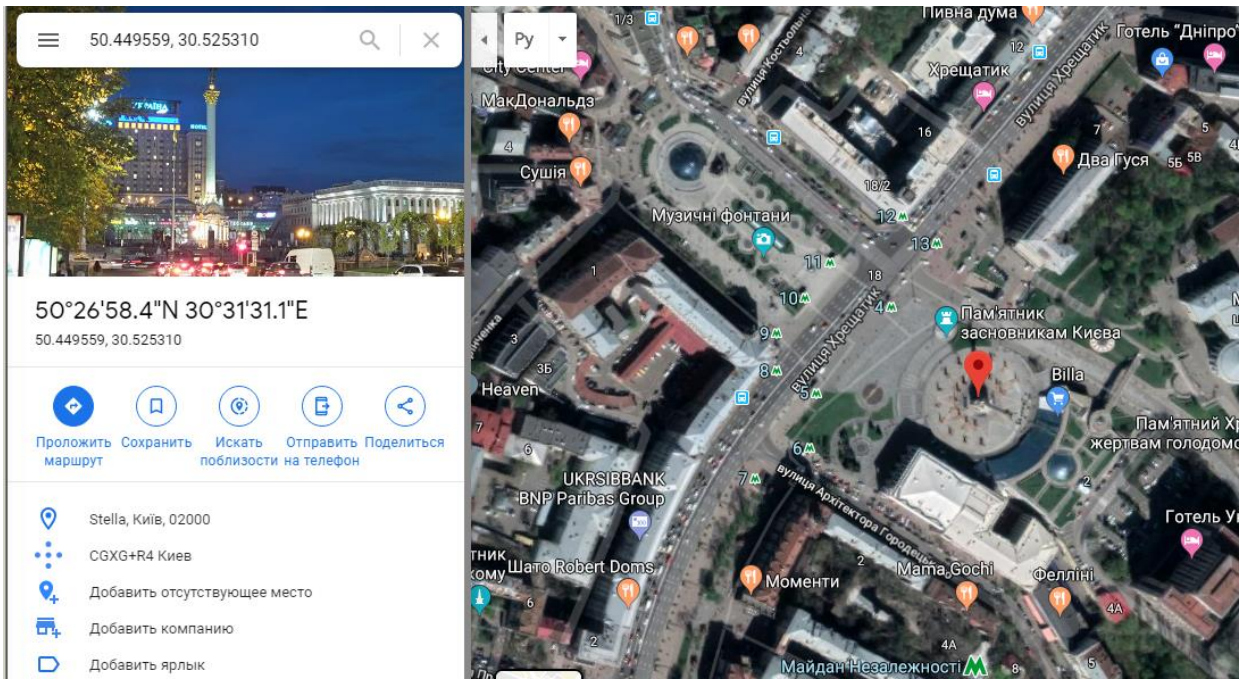


Рис. 1.1. Приклад завдання координат точки на Google-карті.

Як бачимо, координати являють собою записи на зразок дробових чисел із впровадженням додаткових символів типу апострофа, позначки градуса, і т.п. (рис. 1.1, у лівій частині рисунку посередині):

$$50°26’58{,}4’’N \quad 30°31’31{,}1’’E \qquad (1.1)$$

Вказані координати читаються наступним чином: 50 градусів, 26 хвилин, 58.4 секунд північної широти та 30 градусів, 31 хвилина, 31.1 секунда східної довготи.

Ця ж сама точка може мати представлення у десяткових долях градусу (на тому ж рисунку зліва посередині мілким шрифтом):

$$50{,}449559 \quad 30{,}525310 \qquad (1.2)$$

Тут перше число – широта – може мінятися у межах від –90 до 90 градусів (додатні значення відповідають слову «північна»), а друге число – довгота – від –180 до 180 градусів (додатні значення відповідають східній довготі, а від’ємні – західній). Перехід між двома формами запису координат легко виконується, якщо згадати, що у градусі 60 хвилин, а в одній кутовій хвилині міститься 60 кутових секунд, наприклад, від (1.1) до (1.2) перехід буде наступним:

$$50 + 26/60 + 58{,}4/3600 \approx 50{,}449556,$$

$$30 + 31/60 + 31{,}1/3600 \approx 30{,}525306,$$

що майже співпадає з чисельними значеннями (1.2).

Обернений перехід відбувається двома кроками (спочатку обчислюються хвилини, потім – секунди):

[0,449559/(1/60)] = 26’

[(0,449559 – 26/60)/(1/3600)] = 58,4’’

[0, 525306/(1/60)] = 31’

[(0, 525306– 31/60)/(1/3600)] = 31,1’’

Тут, на відміну від попереднього переводу, коли ми отримали лише приблизне співпадіння, бачимо точне співпадіння зі значеннями (1.1) , а це говорить про те, що формат координат виду (1.2) є первинним для системи Google, а у вид (1.1) вони переводяться лише в окремих випадках для «гарного» відображення людині. Форма (1.2) є переважною ще і з міркувань обсягів пам’яті, що необхідні для її зберігання, адже для запису координат саме у цій формі необхідно 20 символів (включаючи роздільну кому та пробіл або символ нового рядка), а для запису у форматі «градуси-хвилини-секунди» потрібно більше (26 символів – з урахуванням усіх роздільних та уточнюючих символів). При великій кількості записів різниця навіть у один символ є суттєвою, тому при однакових усіх інших умовах немає сенсу обирати довшу форму запису, і тому у подальшій роботі будемо розглядати текстовий запис координат тільки у вигляді (1.2).

Найпростішим варіантом збереження будь-якої інформації є її простий безпосередній запис рядком у текстовий файл типу протоколу (логу). Сам лог при цьому виглядатиме так, як на рис. 1.3.

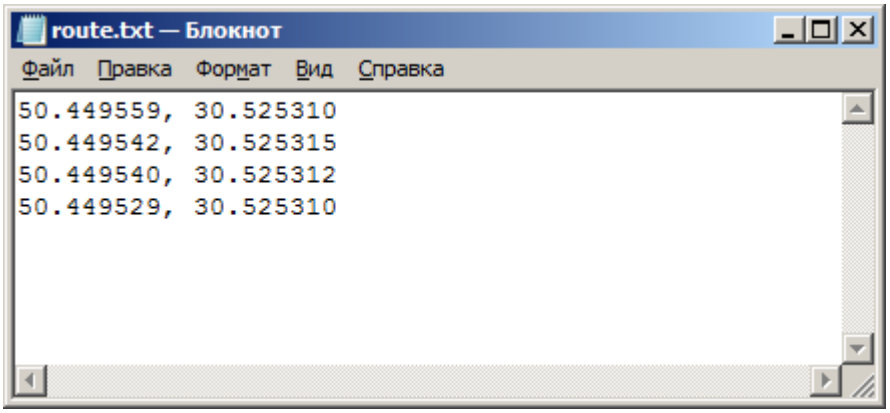


Рис. 1.3. Приклад найпростішого завдання маршруту переміщення.

Самий простий спосіб збереження інформації, нажаль, є самим надлишковим, і, оскільки інформація зберігається на цифрових носіях, не представляє жодних труднощів ввести систему якогось кодування, що могло би значно зменшити обсяг інформації, що підлягає збереженню. Цей підхід аналогічний стисненню інформації без втрат і буде докладніше розглянутий у наступному викладі. Аналогічно методам стиснення інформації із втратами при обробці мультимедійних даних, при збереженні маршрутної інформації також можна викинути частину інформації без втрат якості процесу контролю та запобігання НВРТ. Цей варіант зменшення обсягів архівних даних також буде розглядатися у наступних підрозділах. На даний момент же актуальним є визначення обсягів такої маршрутної інформації у найбільш збитковому, строковому представленні.

Отже, з аналізу рис. 1.1, виразу (1.2) та рис. 1.3 витікає, що для збереження пари координат однієї точки на карті, необхідно 20 символів, тобто при використанні кодування Unicode (2 байти на символ), загальна кількість байтів складатиме:

$$\Delta b = 40 \text{ б.} \qquad (1.3)$$

Тепер слід визначитися, наскільки часто слід зафіксовувати поточні координати. Генерувати маршрутну інформацію можна двома способами з точки зору інтервалу видачі:

- через рівні проміжки часу;

- через рівні ділянки шляху.

Перший варіант є набагато простішим у технічній реалізації, адже без перевірки усіляких додаткових умов (зокрема, без знання та урахування швидкості об'єкта) через заданий проміжок часу відбувається визначення поточних координат і передача їх в архів.

Важливу роль при цьому підході грає вибір інтервалу часу Δt між фіксаціями координат. Оцінити його доцільні значення можна із наступних міркувань:

- швидкість комбайну під час збирання складає від 5 км/год до 15 км/год (наприклад, при збиранні соняшника);

- швидкість комбайну на перегонах максимально може складати біля 30 км/год;

- відстань між послідовними фіксаціями координат для цілей протидії НВРТ може складати порядку 10 м;

- відстань між послідовними фіксаціями координат для цілей запису правильного маршруту на частково пересіченій (або нерівній, і т.п.) місцевості має складати порядку 2 м.

Таким чином, можна виконати дві наступних оцінки для визначення доцільних значень Δt.

а) на перегонах:

Δt_1 = 10 м / 30 км/год = 10 м / 8,3 м/с ≈ 1,2 с.

б) при русі по ділянкам складної геометрії:

Δt_1 = 2 м / 5 км/год = 2 м / 1,4 м/с ≈ 1,4 с.

Як бачимо, у кардинально різних ситуаціях насправді отримуємо досить близькі необхідні часові інтервали між зняттями маршрутної інформації. Вводячи невеликий запас у 20-40%, що цілком оправдано для інженерних задач, можемо прийняти зручну (психологічно та для розрахунків) цифру для Δt на рівні:

$$\Delta t = 1 \text{ с.} \qquad (1.4)$$

Таким чином, для подальшого аналізу способу фіксації положення через рівні проміжки часу приймаємо значення цього проміжку відповідно до (1.4).

При продовженні аналізу, стає очевидним принциповий недолік даного підходу, що надзвичайно ярко проявляється при записі місцезнаходження об’єкта, який нерухомо стоїть протягом тривалого інтервалу часу: якщо, як прийнято вище, положення записувати через кожних Δt = 1 секунд, то за годину (1 год = 3600 с) «стояння» набереться $n_1 = 3600/\Delta t = 3600$ однакових (тобто фактично надлишкових) записів. Враховуючи (1.3), кількість надмірної інформації за годину складе:

$$B_1 = n_1 \cdot \Delta b = 3600 \cdot \Delta b = 3600 \cdot 40 = 144000 \text{ б} = 140 \text{ Кб}.$$

При альтернативному підході, тобто використанні даних про швидкість об’єкта, зокрема тієї обставини (правила), що при нульовій швидкості координати в архіві можна не дублювати, кількість записаних байтів за весь час простою складе просто величину b (1.3).

Протягом робочого дня (вважатимемо тривалість зміни 12 год) при першому підході буде згенеровано:

$$B_{зм} = 12B_1 = 12 \cdot 140 \text{ Кб} = 1680 \text{ Кб} = 1{,}65 \text{ Мб}. \qquad (1.5)$$

З одного боку, ця цифра не є дуже великою, але за сезон загальна кількість інформації для одного комбайну сягатиме сотень мегабайт, а, за умови використання відповідних методів, вона може бути зменшена у кілька разів. Відповідно і об’єми, а отже і кількість, пристроїв для зберігання інформації може зменшуватися, підвищуючи економічну ефективність всієї системи. Таким чином, доцільно проаналізувати весь процес генерації та збереження маршрутної інформації на предмет застосування спеціальних методів, направлених на зменшення об’ємів такої інформації, тож перейдемо до їх розгляду.

1.3. Існуючі способи мінімізації маршрутних даних, що можуть бути застосовані в системах накопичення.

У попередньому підрозділі в загальних рисах описано процес генерації та збереження маршрутних даних, виконано оцінки обсягів інформації, яка при цьому виникає, вказано можливі шляхи удосконалення всього процесу, зокрема застосування методів зменшення обсягу архівних даних без втрат, із втратами, а також метод генерації маршрутних даних, параметри якого залежать від швидкості рухомого об’єкту.

В першу чергу (і це основний спосіб, що застосовується на практиці), маршрутні дані, у якому би форматі вони не зберігалися, можуть бути стиснені без втрат за допомогою звичайних програм-архіваторів. При цьому уже можна досягти значної економії дискового простору, яку можна оцінити за допомогою наступних дій.

Спочатку треба отримати великий обсяг маршрутної інформації, що, за відсутності реальних DATA-файлів можна зробити шляхом інтелектуального моделювання. Згадаємо, що довжина дуги (якою по суті є прямолінійний шлях на поверхні землі) визначається як простий добуток радіуса на центральний кут, що має бути виражений у радіанах:

$$s = \alpha \cdot R \qquad (1.6)$$

Оцінюючи радіус Землі на рівні 6370 км, за (1.6) можна оцінити, що одній мільйонній градуса на поверхні землі відповідає відстань:

$$(0{,}000001^\circ \cdot \pi/180) \cdot 6370000 \approx 0{,}1 \text{ м}, \qquad (1.7)$$

Відповідно, одній стотисячній відповідає відстань біля одного метра:

$$(0{,}00001^\circ \cdot \pi/180) \cdot 6370000 \approx 1 \text{ м}, \qquad (1.8)$$

Відстані 1 км буде відповідати кут біля однієї сотої долі градуса:

$$(0{,}01^\circ \cdot \pi/180) \cdot 6370000 \approx 1000 \text{ м}, \qquad (1.9)$$

Відстані 10 км буде відповідати кут біля однієї десятої долі градуса:

$$(0{,}1^\circ \cdot \pi/180) \cdot 6370000 \approx 10 \text{ км}, \qquad (1.10)$$

Отже, якщо розміри середнього поля прийняти порядку кілометра, то різниця між кутовими координатами для крайніх протилежних його точок

буде складати соті долі градуса. Спираючись на цю інформацію, можна змоделювати великий обсяг маршрутної інформації, наприклад, у середовищі MathCad за наступними залежностями:

$$i := 0..5000000$$

$$Coords_{i,0} := 30.5 + \frac{rnd\left(10^5\right)}{10^6}$$

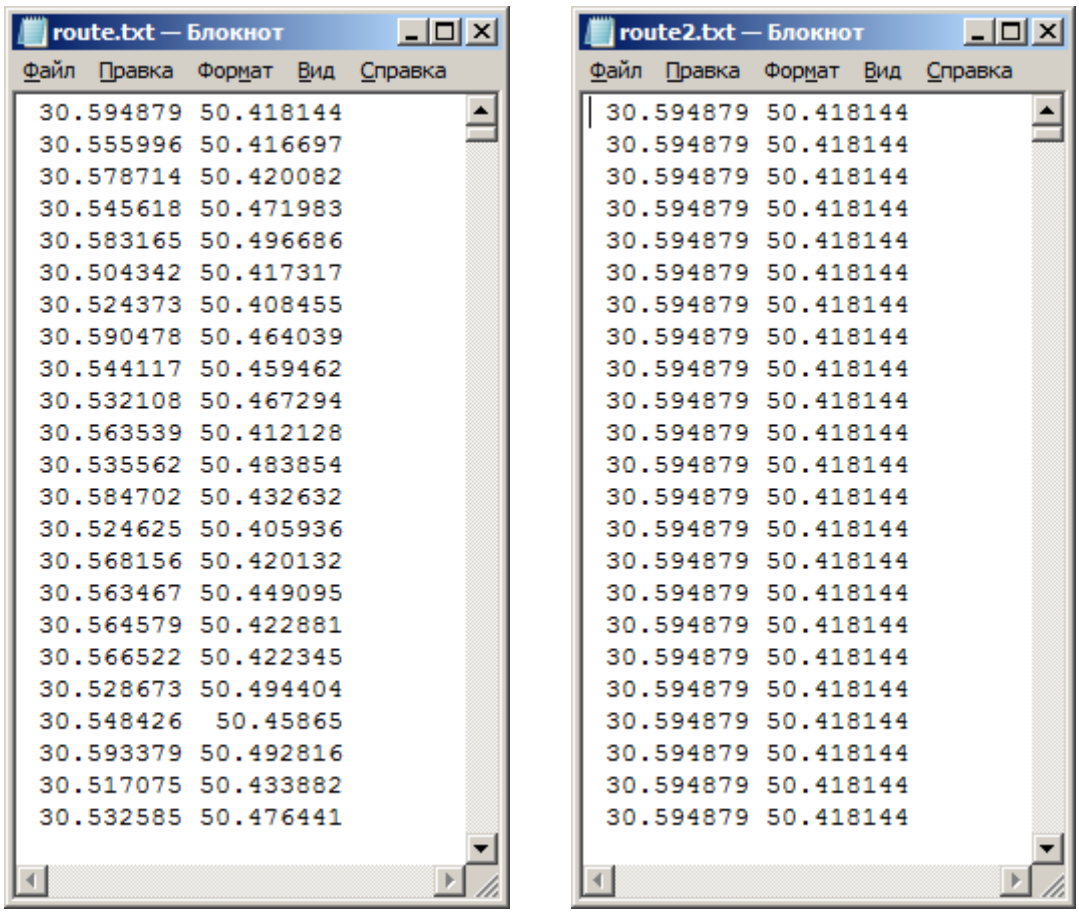
$$Coords_{i,1} := 50.4 + \frac{rnd\left(10^5\right)}{10^6}$$

$$out := WRITEPRN("route.txt", Coords)$$

Тут координати 30,5° східної довготи та 50,4° північної широти прийнято за крайню (ліву нижню) точку поля.

В результаті отримано набір координат, близький до реального – рис. 1.4, *а*. Проведена процедура моделювання є необхідною для правильної оцінки можливого ступеня стиснення файлів з координатами, адже, цей ступінь залежить від структури інформації, що стискається (як приклад, на рис. 1.4, *б* показано файл приблизно того ж розміру, але отриманий шляхом простого копіювання однакових координат – як наочно показано нижче, ступінь стиснення при цьому буде цілком іншою, тому для отримання реального показника можливого ступеня стиснення і треба було провести моделювання).

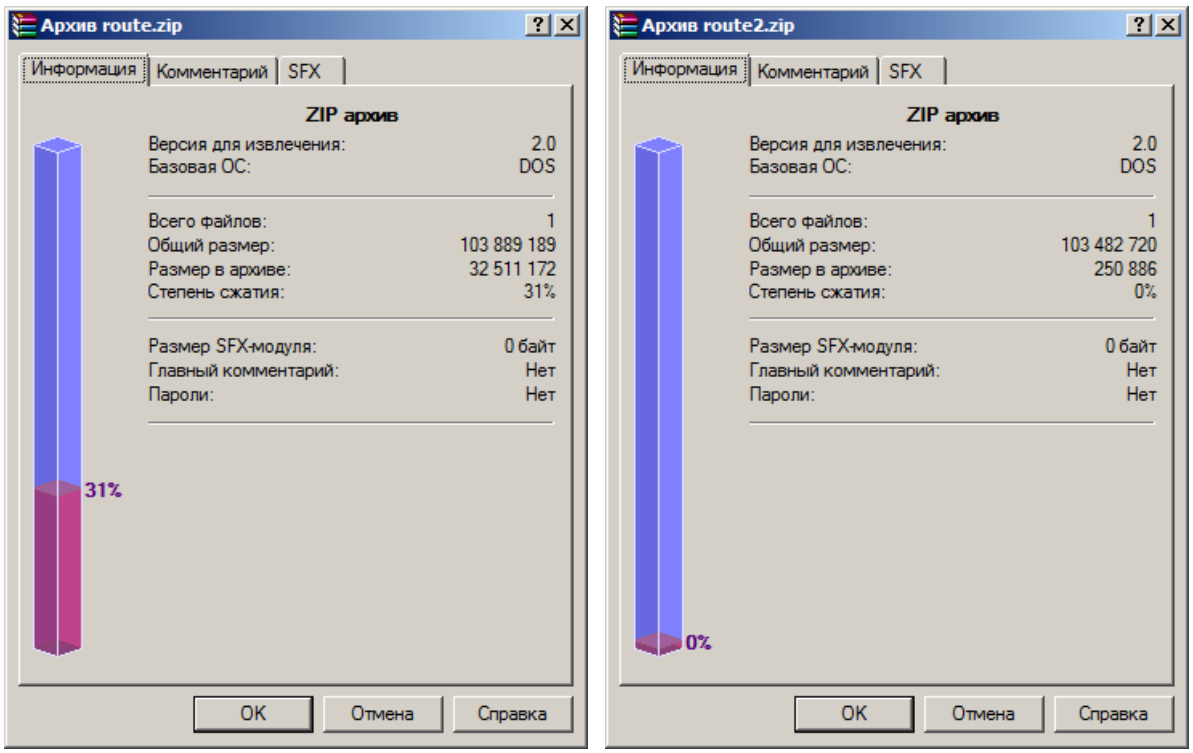
Файл із маршрутною інформацією route.txt було направлено на вхід архіватора, результати роботи якого можна бачити на рис. 1.5, *а*. Видно, що маршрутна інформація, записана у вигляді рис. 1.4, *а* досить добре підлягає стисненню до величини порядку 30% від початкового об’єму (тобто на 70% менше). В той же час на рис. 1.5, *б* показано для порівняння, що стиснення файлу виду 1.4, *б* дає нереальний ступінь стиснення близький до 100%, а це і підтверджує необхідність проведення спеціальної процедури моделювання маршрутної інформації для її подальшого дослідження.



а) б)

Рис. 1.4. Приклади маршрутної інформації: а – змодельована інформація, параметри якої близькі до характеристик реальної маршрутної інформації при переміщенні комбайну по полю; б – файл, отриманий простим копіюванням одного набору координат.

Слід звернути увагу, що розглянуте перетворення інформації являє собою стиснення без втрат (або іншими словами без змін) - архівування, тому що в результаті обернених операцій можна отримати початкову інформацію абсолютно в тій самій формі, що й початково.



а) б)

Рис. 1.5. Приклади стиснення маршрутної інформації простим архівуванням: а – для змодельованої інформації, параметри якої близькі до реальних значень; б – для файлу, отриманого простим копіюванням одного і того ж самого набору координат.

1.4. Аналіз особливостей методів та моделей мінімізації обсягів інформації у залежності від швидкості руху об’єкту.

У попередньому підрозділі розглянуто можливі способи зменшення обсягів маршрутної інформації не прив’язуючись до часових характеристик «поїздки»; передбачається, що координати місцеположення об’єкта знімаються через однакові проміжки часу, незалежно від того, чи рухається об’єкт взагалі, або при більш загальному підході, якою є його швидкість. Однак важливим аспектом цього процесу є якраз залежність параметрів збереження даних від швидкості. Дійсно, очевидним є надлишковість збереження координат через кожну секунду, чи навіть хвилину, якщо ці координати взагалі не змінюються (комбайн припарковано біля ліску).

Таким чином, необхідно ввести певну залежність часу оновлення координат Δt від швидкості руху об’єкта, яка по суті є швидкістю зміни координат об’єкта. Отже, записувати координати слід лише в тому випадку, якщо вони достатньо змінилися. Етап, на якому виконується зменшення інформації через різну швидкість руху, може бути різним:

- безпосередньо при записі;
- при вміщенні маршрутної інформації в архів.

Обидва підходи мають свої плюси та мінуси. Якщо для стиснення застосовується більш-менш інтелектуальний алгоритм, який відповідно потребує значних обчислень, то його впровадження на стороні рухомого пристрою (безпосередньо у комбайні) можливе тільки за умови значного ускладнення відповідної електронної бази, а отже і її подорожчання. Плюсом є те, що для збереження даних на рухомому об’єкті (тобто ще до внесення в централізований архів) потрібно приблизно вдвічі менше обсягів дискового простору (флеш-пам’яті). Економія пам’яті при цьому складатиме не більше 50%. Таким чином, при введенні залежності параметрів маршрутної інформації від швидкості об’єкта безпосередньо на ньому (ще до внесення до архіву) економія досягається за рахунок використання накопичувачів вдвічі меншого обсягу, але більші затрати обумовлені необхідністю встановлення більш повнофункціональних обчислювальних пристроїв (наприклад, процесорів загального призначення замість мікроконтролерів). Зважаючи на те, що для встановлення звичайних процесорів необхідним є ще цілий комплекс апаратного забезпечення (материнська плата, оперативна пам’ять, плати розширення), в такому рішенні додаткові затрати значно перевищать економію.

Таким чином, очевидною є доцільність виконання будь-якої інтелектуальної обробки даних на стороні серверу, безпосередньо перед вміщенням в архів. На рухомих же об’єктах інформація зберігатиметься у нетиснутому в часі вигляді, що абсолютно несуттєво, зважаючи на величезні

місткості сучасних флеш-карт, які доцільно застосовувати для тимчасового локального зберігання інформації.

Так, вартість найпростішої SD-карти обсягом 16 Гб на кінець 2019 року складає біля 4 у.о., а 32 Гб версії – біля 7 у.о. Відносна різниця може видатися великою (майже в два рази), але абсолютна величина різниці 3 у.о. є абсолютно несуттєвою і на порядки (тобто до 100 разів) менше вартості додаткового обладнання, яке треба встановити на кожний комбайн, аби проводити безпосередньо на ньому інтелектуальну обробку даних (стиснення) перед записом у локальну флеш-пам’ять малого об’єму. Таким чином, зважаючи на мізерну вартість флеш-пам’яті, на комбайні достатньо проводити просту фіксацію (запис без стиснення) у флеш-пам’яті поточних координат за допомогою дешевого мікроконтроллера, а потім скидати їх на центральний сервер, де його обчислювальними ресурсами слід проводити стиснення та записувати результат у файловий архів.

Таким чином, навіть при щосекундній фіксації координат протягом доби буде отримано максимум вдвічі більше даних, ніж за (1.5), тобто близько 3,3 Мб. Навіть при обсягу флеш-карти 16 Гб, вона вмістить 5 тис. днів запису, тобто біля 14 років безперервного запису. І повторімося, при перенесенні цієї інформації в архів, де міститимуться дані від сотень (а то і тисяч) комбайнів, питання максимального стиснення стають надзвичайно актуальними і підлягають якомога ефективнішому вирішенню.

Повертаючись до питання залежності координат від швидкості, можна сказати, що вона впроваджуватиметься у кінцевий результат непрямим чином, причому пропозиція конкретного методу буде надана нижче. На даний момент слід лише констатувати, що така залежність має бути урахованою і може надати суттєвої економії необхідного обсягу дискового простору.

1.5. Методи підвищення надійності накопичення даних та їх захисту.

Питання, пов’язані з надійністю та захистом, традиційно включаються у будь-який проект технічної направленості, в тому числі у проекти по розробці програмного забезпечення. Розглянемо тут ці два питання окремо.

Надійність будь-якої системи визначається як її здатність виконувати достатній для користувача обсяг функцій протягом заданого періоду часу. Функції можуть виконуватися у повній мірі – тобто розглядається випадок відсутності будь-яких взагалі відмов. Також часто розглядають часткову втрату функціональності (чи погіршення якості послуг, що надаються системою), тобто коли відмові підлягає одна якась некритична підсистема (компонент), в результаті чого уся система може продовжувати своє функціонування, але не у повному обсязі. Задача інженера-проектанта в частині аналізу надійності якраз і полягає у встановленні важливих типів відмов проектованої системи та визначенні їх числових характеристик (див. нижче у цьому підрозділі).

Для проектованої системи збереження маршрутної інформації доцільно розглядати два типи відмов:

- відмова флеш-пам’яті, яка є найкритичнішою, і яка повністю знищує усі маршрутні дані, що уже було зібрано, а також робить неможливою подальшу роботу всієї системи;

- відмова якого-небудь критичного компоненту системи, що робить подальшу роботу неможливою, але не знищує уже існуючі на флеш-карті дані.

Отже, в подальшому будемо говорити про два цих варіанти відмов, для яких слід розкрити суть надійності.

Відповідно до вищенаведеного, поняття надійності тісно пов’язане із імовірністю відмови системи $Q(t)$ протягом інтервалу часу t. Надійність визначають як імовірність доповнюючої події:

$$P(t) = 1 - Q(t). \qquad (1.11)$$

Іншими словами надійність $P(t)$ є імовірністю того, що за час t від початку своєї роботи, система буде знаходитися у робочому стані.

Для складних систем (якраз таких, як система фіксації та збереження маршрутної інформації), що складаються із кількох (або великої кількості) компонентів, складають схеми надійності, на основі яких по надійностям окремих компонентів визначають загальну інтегральну надійність.

Так, у проектованій системі можна виділити наступні важливі компоненти:

- флеш-пам’ять;
- безпосередньо мікроконтроллер;
- несуча плата;
- GPS-модуль.

Множина критичних пристроїв може позначатися наступним чином:

$\Omega = \{\omega_1 = \text{“флеш”}, \omega_2 = \text{“контроллер”}, \omega_3 = \text{“плата”}, \omega_4 = \text{“GPS”}\}$ (1.12)

Вихід з ладу хоча б одного з цих пристроїв унеможливлює продовження процесу фіксації маршрутної інформації (хоча, якщо компонентом, що відмовив, є не мікросхема флеш-пам’яті, то наявну до моменту відмови маршрутну інформацію можна буде переписати до центрального серверу). Така умова означає, що на схемі надійності усі ці елементи будуть сполучені послідовно – рис. 1.6.

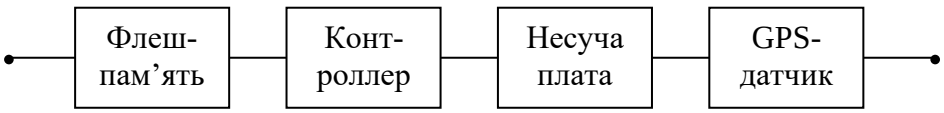


Рис. 1.6. Схема надійності системи накопичення маршрутних даних.

Їхня результуюча надійність, відповідно до схеми рис.1.6 та загальних правил побудови функцій надійності, буде рівною добутку надійностей окремих елементів:

$$P(t) = P_{ф}(t)\cdot P_{мк}(t)\cdot P_{пл}(t)\cdot P_{GPS}(t) = \prod_{i=1}^{4} P_i(t), \qquad (1.13)$$

де $P_{ф}(t)$ – надійність флеш-пам’яті;

$P_{мк}(t)$ – надійність мікроконтроллера;

$P_{пл}(t)$ – надійність несучої плати;

$P_{GPS}(t)$ – надійність GPS-датчика.

Очевидно, кожна із надійностей, що входять до складу (1.13) може оцінюватися за загальноприйнятим експоненційним законом:

$$P_i(t) = \mathrm{e}^{-\lambda_i t}, \tag{1.14}$$

де λ_i – інтенсивність відмов i-того компоненту, яким може бути один із елементів (1.12). Ця величина, як відомо, легко вираховується на основі (є оберненою до) параметру «середній час наробітку на відмову» - MTBF (Mean Time Before Failure), яку порівняно легко можна знайти у довідковій інформації практично для будь-якого електронного пристрою:

$$\lambda_i = \frac{1}{T_i}, \tag{1.15}$$

де T_i – середній час наробітку на відмову для i-того елементу множини (1.12).

Тоді з урахуванням (1.15) та (1.14) вираз (1.13) набуде вигляду:

$$P(t) = \prod_{i=1}^{4} P_i(t) = \prod_{i=1}^{4} e^{-\lambda_i t} = \prod_{i=1}^{4} e^{-\frac{t}{T_i}} = \exp\left(-t \sum_{i=1}^{4} \frac{1}{T_i} \right) \tag{1.16}$$

За цим виразом (1.16) можна розраховувати надійність складної системи із компонентів (1.12), знаючи їх показники MTBF, що часто задаються у довідкових відомостях, слід лише задатися конкретними моделями відповідного обладнання.

Якщо говорити про такий тип відмови, як втрата усіх висхідних маршрутних даних, то в такому випадку ланцюг рис. 1.16 вироджується в одну єдину комірку (надійність флеш-пам’яті), а отже і загальна надійність системи збереження рівна надійності цього елемента:

$$P(t) = P_{ф}(t).$$

Таким чином, в рамках стандартної теорії надійності розглянуто питання забезпечення відмовостійкості розроблюваної системи. Що ж стосується питань захисту інформації, то, як відомо, усі засоби захисту можна поділити на три типи, залежно від типу загрози, на нейтралізацію якої вони направлені:

- засоби забезпечення цілісності;
- засоби забезпечення конфіденційності;
- засоби забезпечення доступності.

В даному випадку, найбільшу увагу очевидно слід приділяти засобам забезпечення цілісності маршрутної інформації, оскільки, нажаль, та особа, що тривалий час знаходиться поруч із важливою інформацією (оператор сільськогосподарської техніки), якраз і несе для неї найбільшу загрозу (може стерти, знищити або пошкодити носій, і т.п.).

Загрози конфіденційності маршрутної інформації не можуть бути встановлені через практично їх повну відсутність, рівно як і загрози доступності (якщо виключити малоймовірне тимчасове пошкодження входу слоту для встановлення флеш-пам’яті із збереженням її вмісту). Таким чином, основна увага в частині захисту інформації має бути приділена питанням забезпечення цілісності інформації.

Як відомо, існують три типи заходів для вирішення проблем ЗІ:

- правові;
- організаційні;
- технічні.

Правові заходи не можуть вирішити проблему забезпечення цілісності, оскільки і так є очевидним, що знищувати обладнання чи пошкоджувати його окремі компоненти будь-кому (в першу чергу, працівникам компанії) заборонено. Технічні заходи також не підходять для вирішення поставленої задачі через їх високу вартість (адже практично будь-які технічні рішення потребують розробки проекту, реалізації дослідних зразків, тестування, впровадження та витрат на експлуатацію). Таким чином, оптимальним

рішенням у даному випадку є впровадження засобів організаційного характеру. Конкретніше, необхідно:

- пристрій для збереження маршрутної інформації (мікроконтроллер, GPS-модуль, флеш-пам’ять) слід розмістити окремо від інших засобів керування електронними компонентами рухомої техніки;

- пристрій необхідно захистити надійним корпусом (бажано, звичайною металевою пластиною/пластинами);

- слід убезпечити підвід електроживлення до пристрою (можливо, навіть початково обираючи місце розташування пристрою якомога ближче до виводів системи електропостачання);

- виймання заповнених флеш-карт та передачу їх до дата-центру, а також доставку порожніх карт та їх встановлення, слід виконувати за допомогою спеціально призначеного працівника;

- за умови спорядження пристрою GSM-модулем, процедура зчитування флеш-карт має бути повністю автоматичною, прозорою для оператора;

- в жодному разі, за жодних умов оператор не повинен мати доступу до флеш-карти чи елементів управління пристрою зняття маршрутної інформації.

Притримуючись описаних простих принципів, можливо забезпечити достатній рівень надійності та захисту інформації в системі збереження маршрутної інформації.

1.6. Висновки по розділу.

У розділі розглянуто основні відомості про методи та засоби вирішення проблеми збереження маршрутної інформації, що існують на сьогоднішній день у відповідній галузі. Спочатку проаналізовано варіанти застосування маршрутної інформації у різноманітних системах, зокрема агротехнічних. Проведено моделювання процесу генерації маршрутних даних у середовищі Mathcad, що необхідно для реалістичної оцінки ступеня стисливості

маршрутної інформації. Встановлено, що інформація у безпосередньому, непідготовленому вигляді може бути стиснена до близько 30% від початкової величини.

У розділі розглянуто способи зняття маршрутної інформації, проаналізовано її структуру та особливості, намічено шляхи для можливих варіантів її додаткового стиснення. Також у розділі розглядаються питання залежності параметрів маршрутної інформації від швидкості об’єкта, а також питання надійності та захисту інформації.

В цілому, встановлено, що існують шляхи покращення способів стиснення маршрутної інформації, які є доцільним розробляти у наступних розділах даної роботи.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ МІНІМІЗАЦІЇ МАРШРУТНИХ ДАНИХ ДЛЯ СИСТЕМ НАКОПИЧЕННЯ

2.1. Обґрунтування вибору базового методу мінімізації маршрутних даних.

Для системи, що розглядається, можуть бути застосовані дуже і дуже різноманітні методи мінімізації маршрутних даних. Розглянемо головні особливості основних із них.

В першу чергу, можливим є застосування загальних методів стиснення інформації без втрат – архівування. Аналіз його особливостей та шляхів удосконалення методі архівування виходить далеко за рамки даної роботи і являє собою величезний пласт знань з галузі теорії інформації. Таким чином, у даному дослідженні архівування будемо застосовувати відповідно до стандартних методів (наприклад, по алгоритму ZIP), причому в останню чергу, коли усі інші методи стиснення уже використано.

По-друге, можна розглядати методи, які беруть до уваги фізичну суть даних, що підлягають стисненню. А саме, практично усі (крім деяких службових відомостей) числа, які наявні у файлі з маршрутною інформацією, є парами координат точок земної поверхні, причому точок сусідніх. З окремих послідовно розміщених точок формується ламана лінія, яка, при достатній щільності вузлових точок, може досить точно представлятися кривою, сплайном. Отже, постає задача: по набору послідовно розміщених точок отримати коефіцієнти сплайну відповідного степеня, причому набір цих коефіцієнтів має займати меншу кількість байтів, ніж набір координат точок, по яким побудовано сплайн.

Згадаємо, що координати двох точок точно «лягають» на пряму лінію, рівняння якої складається з двох коефіцієнтів ($y = ax + b$, два коефіцієнти a та b). Три точки точно розміщуються на кривій другого порядку - звичайній параболі, що має три коефіцієнти ($y = ax^2 + bx + c$, три коефіцієнти a, b, c). Так і далі: для точного розміщення n точок на кривій n-ного порядку потрібно рівно n коефіцієнтів многочлену n-ного степеня:

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_2 x^2 + a_1 x + a_0 = \sum_{k=0}^{n} a_k x^k \qquad (2.1)$$

Таким чином, якщо використовувати рівняння многочлену *n*-ного порядку, то зберігати необхідно буде лише координати x_i, а усі значення y_i (яких всього *n* штук) будуть розраховуватися по формулі (2.1) і їх зберігати більше не потрібно. Але, окрім усіх x_i зберіганню підлягають усі коефіцієнти a_k виразу (2.1), яких всього також *n* штук. Таким чином, якщо точність завдання коефіцієнтів a_k не нижче точності завдання y_i, то жодної економії від такої заміни не спостерігатиметься. У протилежному випадку (при зменшенні точності зберігання коефіцієнтів a_k), зважаючи на те, що на основі цих коефіцієнтів арифметичним шляхом вираховуються значення y_i, то і ці результуючі значення будуть визначатися неточно (оскільки в (2.1) коефіцієнти входять у перших степенях, то і неточність визначення розрядів зберігатиметься: наприклад, якщо якийсь коефіцієнт a_k має неточність у другому розряді після коми, то і загальний результат, тобто y_i, матиме таку ж неточність у другому розряді). Зважаючи на загальну велику кількість коефіцієнтів a_k та можливість існування неточностей досить великих порядків у кожного з них, загальні похибки визначення координат y_i за (2.1) можуть спотворити маршрут так, що користуватися ним (переглядати, перевіряти і т.п.) буде абсолютно неможливо.

Таким чином, переходити до сплайн-апроксимації в задачі збереження маршрутної інформації не доцільно. Розглянемо інший, більш універсальний спосіб перетворення інформації, що може бути корисним для цілей даного дослідження.

Як відомо, окрім множини дійсних чисел, що можуть використовуватися для вимірювання фізичних величин, існує цілий клас комплексних чисел, в яких бере участь спеціальна комплексна одиниця $i = \sqrt{-1}$. Через уявний характер цього символу *i*, комплексні числа не можна використовувати для опису реальних явищ. Однак, такі числа знайшли

широке застосування в задачах перетворення сигналів та у більш широкому сенсі – інформації.

Суть численних «комплексних» перетворень (їх називають перетвореннями Фур'є, хоча історично першою назва «перетворення Фур'є» була використана для перетворення на основі тригонометричних функцій синус та косинус) полягає у наявності двох правил:

- як на основі набору дійсних чисел отримати відповідний їм набір комплексних чисел;

- як на основі набору комплексних чисел повернутися до початкового набору дійсних чисел.

Перетворення Фур'є доцільно виконувати через ту обставину, що деякі речі, що дуже складно виконуються над дійсними числами (наприклад, такою операцією є процес вирішення диференціального рівняння), набагато легше робляться із їх комплексними образами (диференціальне рівняння заміняється алгебраїчним, що завжди вирішується значно простіше). Отже, корисна схема застосування інтегральних перетворень типу Фур'є може бути наступною:

а) виконати комплексне перетворення висхідного дійсного сигналу $f_1(t)$ (інформації), або, як кажуть, оригіналу, і отримати комплексний вираз $F_1(p)$, що йому відповідає, або зображення:

$$F_1(p) = \int_T K(p,t)\, f_1(t) dt, \qquad (2.2)$$

де $K(p, t)$ називають ядром інтегрального перетворення;

T – область інтегрування у дійсній області, прийнята для даного ядра $K(p, t)$.

б) виконати над комплексним образом $F_1(p)$ перетвореного вхідного сигналу необхідні дії, направлені на вирішення початкової задачі по обробці оригінального вхідного сигналу і отримати змінений комплексний образ $F_2(p)$;

в) виконати обернене комплексне перетворення, яке по зміненому комплексному образу $F_2(p)$ відтворить потрібний вихідний сигнал $f_2(t)$ у звичайній, дійсній формі:

$$f_2(p) = \int_P K^{-1}(p,t) F_2(p) dp . \qquad (2.3)$$

Таким чином, шлях виконання операцій

$$f_1(t) \to F_1(p) \to F_2(p) \to f_2(t) \qquad (2.4)$$

при застосуванні інтегральних перетворень типу Фур'є є значно легшим ніж пряма переробка інформації $f_1(t) \to f_2(t)$.

Теоретично, схема (2.4) може стати в нагоді і для організації системи стиснення маршрутної інформації, але тільки при виконанні наступних умов:

а) якщо перетворений сигнал $F_1(p)$ можна якимось способом зменшити в об'ємі, отримавши змінений вираз-зображення $F_2(p)$;

б) якщо розмір $F_2(p)$ у пам'яті ЕОМ став меншим за розмір висхідного дійсного сигналу $f_1(t)$. Іншими словами має виконуватися:

$$sizeof(F_2(p)) < \text{sizeof}(f_1(t)), \qquad (2.5)$$

де оператором *sizeof*() позначено взяття розміру у байтах. Якщо (2.5) не виконується, то у всьому перетворенні просто немає сенсу, оскільки ніяке стиснення не відбувається;

в) якщо по зміненому зображенню $F_2(p)$ можна відновити сигнал $f_2(t)$, який, взагалі кажучи, буде відрізнятися від початкового сигналу $f_1(t)$, так як у них різні зображення $F_1(p)$ і $F_2(p)$. Тим не менше, ці два сигнали мають бути близькі один до одного. Іншими словами:

$$diff\,(f_1(t), f_2(t)) < \Delta f, \qquad (2.6)$$

де оператором *diff* позначена операція знаходження різниці між двома сигналами, а Δf – порогове допустиме значення цієї різниці. У якості конкретної операції знаходження різниці можна взяти інтегрування арифметичної різниці двох функцій по їх області визначення:

$$diff\,(f_1(t), f_2(t)) \overset{def}{=} \frac{1}{T}\int_T \left| f_1(t) - f_2(t) \right| dt . \qquad (2.7)$$

Для визначення можливості застосування цієї схеми стиснення у реальній системі збереження маршрутної інформації слід проаналізувати її потенційну ефективність. Для цього задамося стандартним перетворенням Фур'є, в якому ядро має вид $K(p, t) = e^{-pt}$, та реалізуємо інтегральне перетворення у системі Mathcad, яка уже вище використовувалася.

Це програмне середовище містить функцію fft(), що здійснює «швидке» перетворення Фур'є, особливістю якого є те, що вхідний вектор чисел повинен містити 2^m елементів (тобто їх кількість має бути степенем двійки). Зважаючи на те, що потік координат маршрутної інформації є неперервним, не представляє особливої проблеми його розбиття на частини по 2^m чисел. При цьому на виході функція fft() дає 2^m–1 комплексних чисел тієї ж точності, а оскільки кожне комплексне число є парою двох дійсних чисел, то загальна кількість інформації при такому перетворенні ($f_1(t) \rightarrow F_1(p)$) не змінюється. В той же час зображення $F_1(p)$ можна стиснути, реалізувавши перетворення $F_1(p) \rightarrow F_2(p)$. Наприклад, це можливо, якщо оцінювати модуль кожного комплексного числа та числа з малими модулями, меншими граничного значення ΔF (тобто такі, що вносять у загальний результат дуже малий внесок) просто обнуляти (зменшуючи таким чином, кількість інформації у $F_2(p)$). Після виконання такої процедури стиснуте зображення $F_2(p)$ можна піддати оберненому перетворенню Фур'є і отримати новий оригінал $f_2(t)$, який, як уже зазначалося вище, буде трохи відрізнятися від висхідного сигналу $f_1(t)$. Задача тут полягає в оцінці, як вибір порогу ΔF впливає на різницю (2.4) між двома дійсними сигналами, що виникає при певному ΔF.

Для дослідження окреслених питань реалізовано прості функції користувача системи MathCad, код яких наводиться нижче:

а) функція compressZero(fCoords, ΔF), яка обнуляє усі комплексні числа в матриці fCoords, модуль яких менше заданого порогу ΔF:

$$\text{compressZero}(A, \Delta F) := \left| \begin{array}{l} \text{for } i \in 0..\text{length}(A) - 1 \\ \quad A_i \leftarrow 0 \text{ if } \sqrt{\left(\text{Re}(A_i)\cdot\text{Re}(A_i) + \text{Im}(A_i)\cdot\text{Im}(A_i)\right)} < \Delta F \\ A \end{array} \right.$$

б) функція NumZeroed(fCoords, ΔF), яка рахує кількість комплексних чисел у матриці fCoords, модуль яких менше заданого порогу ΔF:

$$\text{NumZeroed}(A, \Delta F) := \left| \begin{array}{l} n \leftarrow 0 \\ \text{for } i \in 0..\text{length}(A) - 1 \\ \quad n \leftarrow n + 1 \text{ if } \sqrt{\left(\text{Re}(A_i)\cdot\text{Re}(A_i) + \text{Im}(A_i)\cdot\text{Im}(A_i)\right)} < \Delta F \\ n \end{array} \right.$$

Після застосування цих функцій (приклад застосування наведено на рис. 2.1) отримуємо наступні результати для різних величин порогів ΔF, які зведено в табл. 2.1.

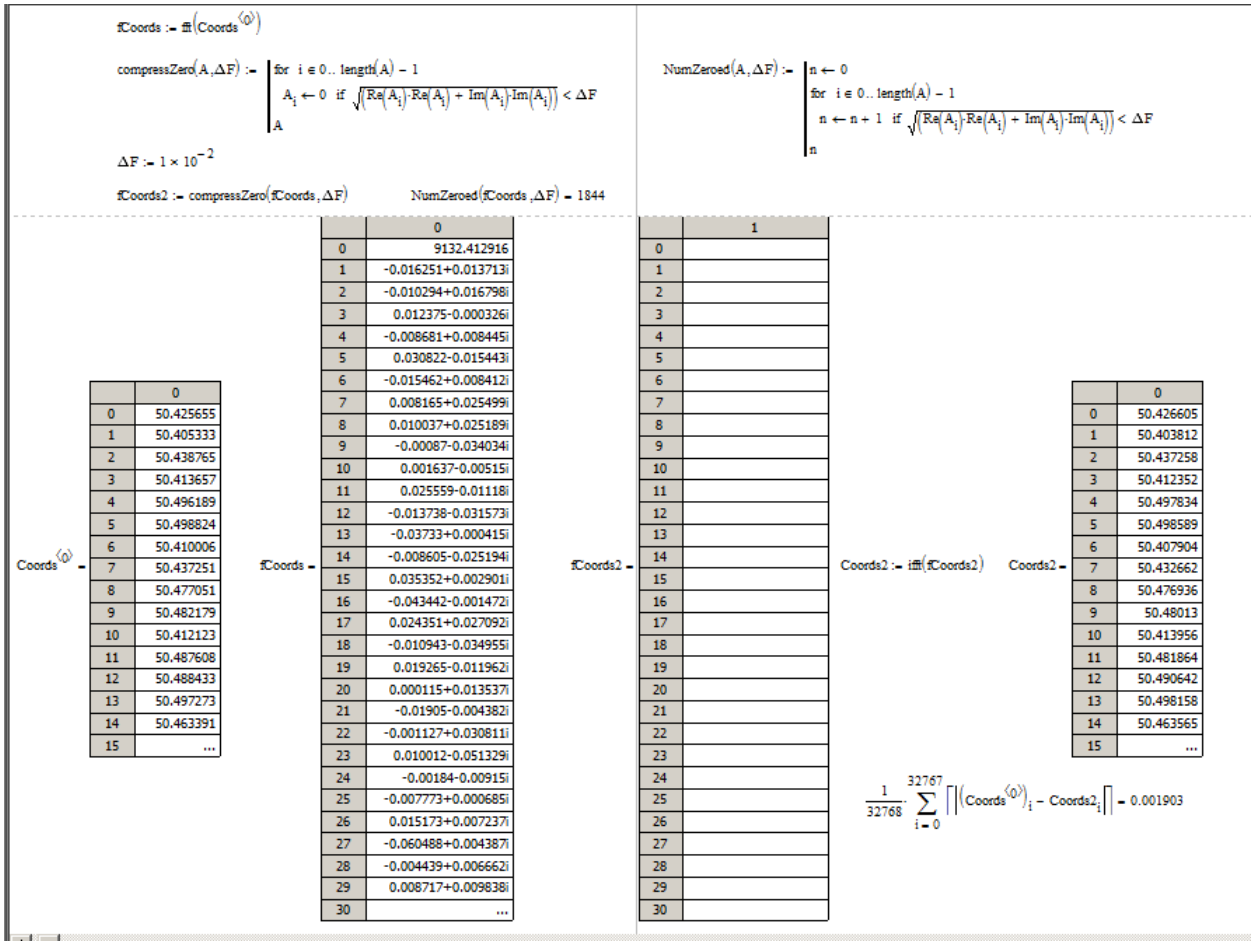


Рис. 2.1. Приклад застосування інтегрального перетворення для стиснення маршрутної інформації (для значення порогу $\Delta F = 10^{-2}$).

Табл. 2.1. Параметри системи стиснення маршрутних даних залежно від значень порогового значення ΔF.

№	ΔF	Обсяг фрейму, відліків	Занулених фреймів	На скільки зменшується обсяг, %	Середнє відхилення, diff, градусів	Середнє відхилення, метрів
1.	$0{,}5\cdot10^{-2}$	32768	460	1,4	0,0005	50
2.	10^{-2}	32768	1821	5,5	0,002	200
3.	$2\cdot10^{-2}$	32768	6238	19	0,007	700
4.	$3\cdot10^{-2}$	32768	10797	33	0,013	1300

Аналіз табл. 2.1, нажаль, показує, що застосовувати ефективне стиснення на основі інтегральних перетворень по схемі (2.4) при умові виконання (2.6)-(2.7), тобто при забезпеченні достатньої якості відновленого сигналу, не представляється можливим.

Дійсно, у першому рядку показано параметри варіанту розрахунків із порогом $\Delta F = 0{,}5\cdot10^{-2}$, при якому відкиданню підлягають 460 комплексних чисел, тобто близько 1,4%. Середнє відхилення однієї точки при оберненому перетворенні, тобто при відтворенні $f_2(t)$, складає біля 50 м, що знаходиться на грані точності завдання маршрутної інформації (навіть трохи виходить за цю грань, оскільки в умовах пересіченої місцевості випадкове відхилення точки величиною 50 метрів повністю спотворює зміст маршрутної інформації). Одним словом, середнє відхилення зафіксованих точок маршруту, що виникає при стисненні за допомогою інтегрального перетворення, має бути меншим, тобто поріг ΔF має бути зменшений. Однак, навіть при значенні $\Delta F = 0{,}5\cdot10^{-2}$, що розглядається, обсяг інформації зменшується на всього лише біля 1%, а якщо поріг зменшити, то і обсяг інформації, що відкидається, стане ще меншим одного відсотка, що абсолютно неефективно для задач стиснення даних.

Підсумовуючи, можна сказати, що ані метод сплайн-апроксимації, ані методи інтегрального перетворення Фур'є не дають ефективних результатів для задачі стиснення маршрутної інформації перед її вміщенням в архіви. В той же час, беручи до уваги відомості, наведені у першому розділі, можна зробити висновок, що найоптимальнішим способом обробки маршрутної інформації є розгляд та урахування самих її властивостей та стиснення на основі їх аналізу, без необхідності складної математичної обробки.

2.2. Аналіз можливостей удосконалення обраного методу мінімізації маршрутних даних.

Отже, мінімізацію будемо проводити на основі аналізу властивостей маршрутної інформації порівняно простими з математичної точки зору методами.

В першу чергу, слід звернути увагу, що значення координат, які надаються GPS-датчиком, записані у долях градусу і містять мільйонні його долі (див. напр. (1.2)). У розрахунку (1.7) було встановлено, що одній мільйонній долі градуса приблизно відповідає відстань 0,1 м, що абсолютно не потрібно для предметної галузі, яка розглядається (точність зависока). Це означає, що цілком без втрат важливої змістової інформації можна викинути мільйонні цифри (розряди), обмежившись стотисячними долями градусу, що дають відстані між двома послідовними положеннями порядку метра. Таке зменшення кількості розрядів, що відображаються, для обох координат надасть зменшення обсягів інформації, що зберігається, приблизно на $1/8 \approx 12\%$, що теж не мало. Подальше відкидання розрядів вже не є обґрунтованим, оскільки сприятиме виникненню помилок у визначенні місцеположення на цілі метри відповідно до (1.8), що може бути критичним для опису особливостей маршруту, наприклад, на частково пересіченій місцевості.

Якщо подивитися на координати з іншого боку (зі сторони вищих розрядів), то можна бачити повторення іншого типу, коли повторюється

значення власне градусів, а також їх вищих розрядів десяткових дробів (наприклад, на рис. 1.4, *а* усі записи довготи починаються із цифр 30.5, а широти починаються із цифр 50.4). Очевидно, що наявність таких повторів також можна ефективно використовувати для стиснення маршрутної інформації. Наприклад, використовуючи тег BASE (або з метою економії просто B), можна задавати базові значення довготи та широти, до яких будуть додаватися ті значення, які масово вказуються у рядках нижче. Наступний тег типу BASE буде зустрічатися, коли зміниться та частина координати, що прописана у цьому тегу.

Наприклад, при роботі в рамках одного сільськогосподарського поля, різниця між кутовими переміщеннями по довготі та широті не буде перевищувати 0,1°, оскільки за (1.10) така кутова відстань відповідає лінійному переміщенню близько 10 км. Навіть, якщо зміна кутових координат і буде відбуватися у межах 0,1 градуса, це буде відбуватися надзвичайно рідко і буде перекриватися введенням тегу типу BASE.

Ще більш гнучкий підхід можна впровадити, розглядаючи окремо два базових теги:

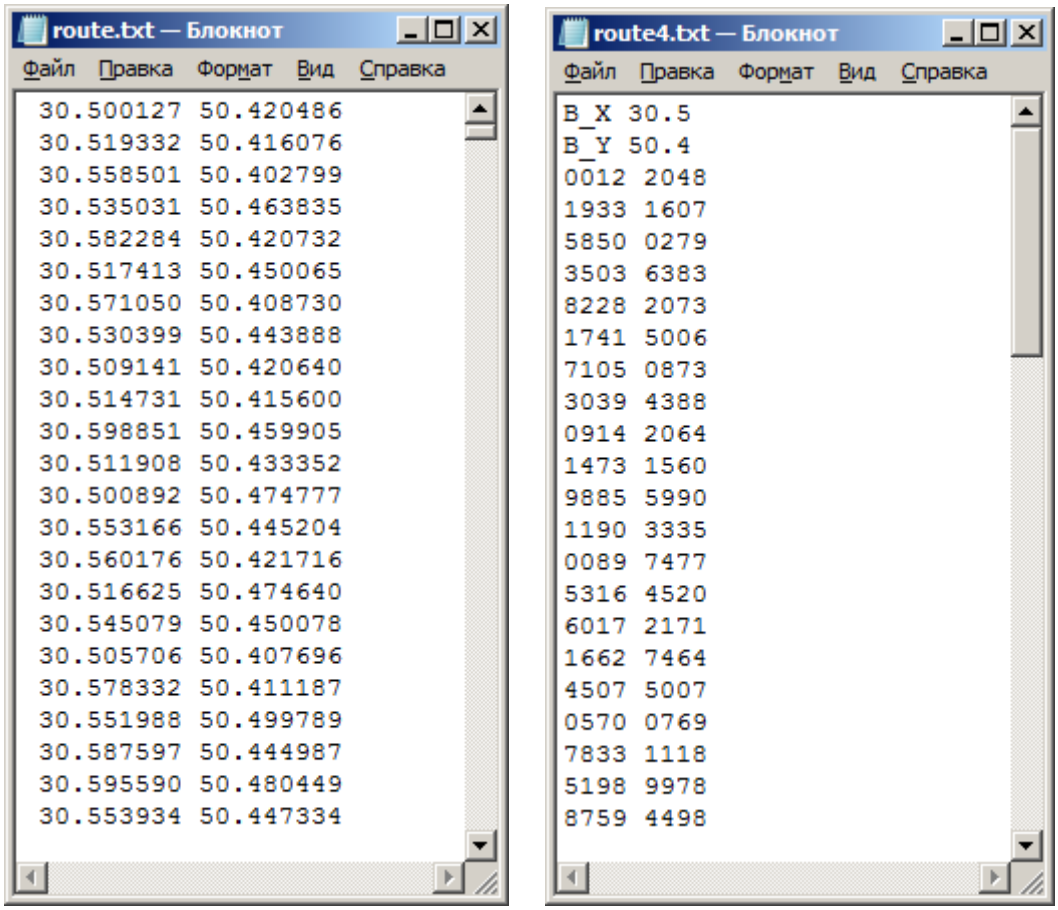
а) B_X – для зміни базової частини довготи;

б) B_Y – для зміни базової частини широти.

При цьому фрагмент маршрутної інформації буде виглядати як на рис. 2.2, *б*. Для порівняння на рис. 2.2, *а* показано фрагмент того ж файлу до відкидання зайвих мільйонних долей градусу та кодування базової частини координат.

При такому перетворенні зменшення обсягів інформації відбувається приблизно вдвічі, причому фактично без втрат, тому що виконане відкидання мільйонних долей градусу є несуттєвим для предметної галузі, яка розглядається. Для повноти картини слід відмітити, що ступінь стиснення при архівуванні нового формату як на рис 2.2, *б* стає дещо меншим, ніж як на рис. 2.2, *а* (близько 45-50%), але все одно невелике зменшення ступеня стиснення архіватором (з 45% до 30%) компенсується великим зменшенням

обсягу висхідних даних (приблизно в два рази) і в цілому ступінь стиснення інформації при застосуванні обох методів разом є більшою, ніж при застосуванні кожного з них окремо.



а) *б*)

Рис. 2.2. Перетворення координатної інформації: а – оригінал, б – та ж сама інформація після відкидання мільйонних часток та введення базової частини кожної координати.

Окрім вищевказаних дій, наступний крок стиснення може бути реалізований через переведення кожної чотиризначної координати (рис. 2.2, *б*) у числову форму. Оскільки цифр всього 4, то кожне таке число може вміщуватися у 2-байтову змінну типу short int. Отже, замість чотирьох байтів, що відводяться у текстовому файлі під зберігання «слова» із 4 цифр, у кінцевому стисненому форматі на кожну координату відводитиметься

2 байти, за рахунок чого весь обсяг інформації зменшується майже вдвічі (за винятком службової інформації типу тегів BASE).

Тут слід звернути увагу на те, чому до бази були включені десяткові долі координат (окрім самих координат, звичайно), а соті, тисячні, десятитисячні та стотисячні долі залишаються і формують вказані чотиризначні числа. Задача вибору розрядів, що включалися би до бази, є класичною задачею оптимізації із конфліктуючими вимогами.

По-перше, слід намагатися, щоби службові записи типу слова BASE (чи B_X та B_Y) зустрічалися якомога рідше, оскільки велика їх кількість збільшуватиме загальний обсяг файлу. Отже, до них слід намагатися включити якомога менше розрядів, наприклад, самі координати, без дробової частини взагалі. Однак, при цьому для кожного запису місцеположення об’єкта треба буде зберігати цілих 5 цифр, тобто максимально – число 99999, для розміщення якого у пам’яті ЕОМ потрібно більше, ніж 2 байти. Оскільки створити змінну розміром 3 байти стандартними способами неможливо (а використання такої структури/запису буде вкрай складним технічно), то реально для таких чисел слід використовувати 4-байтові змінні типу int. Таким чином, економлячи на тегах типу BASE, більше втрачаємо на представленні кожної координати. Якщо ж до бази включити окрім цілих значень, ще і десяткові розряди, то тег BASE буде зустрічатися трохи частіше, однак для кожної координати залишиться 4 розряди, тобто максимальне число 9999, яке вміщується у 2-байтову змінну типу short int. Таким чином, досягається максимальна економія дискового простору при зберігання цілого файлу. Якщо далі збільшувати кількість розрядів, що включатимуться до бази, то при включенні сотих долей градусу тег BASE буде зустрічатися ще частіше, однак на кожну координату все одно треба буде відводити 2 байти (аби зберігати у таких змінних числа від 000 до 999, що не вміщуються в один байт). Таким чином, цей варіант однозначно гірший за попередній. Останнім варіантом, який доцільно розглянути, є включення до бази ще й тисячних долей градусу, при яких на кожну

координату залишається всього лише два розряди (тобто це числа від 00 до 99), які можна вмістити в один байт. Однак, при зміні навіть тисячної долі градусу слід буде вводити окремий тег типу BASE, а 0,001° відповідає лінійному переміщенню по поверхні Землі приблизно на 100 м. Таким чином, якщо сусідні точки маршруту розміщуються набагато ближче одна до одної, ніж 100 м, то доцільним є введення до тегу BASE ще і тисячних долей градусу, а якщо сусідні точки розміщені на відстанях порядку десятків метрів (тобто тисячні долі градусу змінюються досить часто), то BASE доцільно обмежувати десятими долями градусу. Уся ця інформація у зведеному вигляді наведена в табл. 2.2. В таблиці наведено половинчасті значення середніх лінійних відстаней, на які слід зрушити з місця, щоби змінилася відповідна база, оскільки рух можливий як в одну, так і в іншу сторони.

Табл. 2.2. Аналіз ефективності розподілу розрядів координат між базою та окремими записами (радіус Землі узято рівним 6370 км).

№	Число дробових розрядів у базі	Відстань на яку слід зрушити, щоби змінилася база, °, м	Число розрядів у кожній окремій координаті	Об’єм кожної окремої координати, байт	Доцільність використання на практиці
1.	0, числа виду XX	0,5°, 55589 м	5	4	–
2.	1, числа виду XX,X	0,05°, 5559 м	4	2	+
3.	2, числа виду XX,XX	0,005°, 556 м	3	2	–
4.	3, числа виду XX,XXX	0,0005°, 56 м	2	1	+
5.	4, числа виду XX,XXXX	0,00005°, 6 м	1	1	–

Таким чином, підсумовуючи питання розподілу розрядів координат між базою та окремими записами, можна сказати, що для цілей фіксації маршрутної інформації сільськогосподарської техніки найкраще підходить варіант №4.

Усі наведені удосконалення є відносно незалежними, тому можуть бути застосовані усі разом, в комплексі, формуючи відповідний алгоритм, що і буде реалізовано у даній роботі, а докладно розглянуто у наступному підрозділі.

2.3. Розробка алгоритмів, пов'язаних із мінімізацією маршрутних даних.

Таким чином, у попередньому підрозділі запропоновано декілька операцій щодо стиснення маршрутної інформації, яка надається GPS-датчиком, на основі яких може бути сформований алгоритм, що складається з наступних кроків:

1) відкидання мільйонних часток (шоста цифра після десяткової коми) в *усіх* наявних координатах;

2) вибір кількості десяткових розрядів (варіант 2 або 4 із табл. 2.2), що включатимуться до бази, та, відповідно, інших, що залишатимуться у кожній координаті (залежно від характеру маршрутної інформації; для випадку руху сільськогосподарської техніки по полю оптимальним є варіант 4 із табл. 2.2);

3) формування міток типу BASE, в яких задається базова (незмінна в рамках великого блоку даних) початкова частина кожної координати, та відкидання цієї базової частини у межах одного блоку в усіх наявних у блоці координатах;

4) виконання кодування кожного чотиризначного числа, що відповідає одній із координат, однією числовою змінною типу short int – за умови, що у кроці 2 було обрано варіант №2 із табл. 2.2;

5) виконання кодування кожного двохзначного числа, що відповідає одній із координат, однією числовою змінною типу char/byte – за умови, що у кроці 2 було обрано варіант №4 із табл. 2.2;

6) архівування отриманого файлу з координатами за допомогою архіватора.

Схема цього алгоритму наведена на рис. 2.3.

Отриманий файл слід розмістити у централізованому сховищі (на одному виділеному сервері), а, при виникненні необхідності, спеціалізоване програмне забезпечення, що підлягає розробці в рамках даної роботи, повинно надавати можливість користувачеві здійснити обернене перетворення і отримати маршрутну інформацію у зрозумілій (текстовій) формі, що може розпізнаватися системами типу Google Maps.

Для тестування цього алгоритму на вхід має бути поданою реальна, або наближена до неї по своїм параметрам інформація. У підрозділі 1.3 уже виконувалося моделювання маршрутної інформації, однак із зовсім іншою ціллю – оцінити ступінь її стисливості звичайними архіваторами, тому наведений там елементарний спосіб генерації був цілком задовільним. Зараз же необхідно в тому числі візуалізувати маршрут наочно, тому наведена раніше процедура створення випадкових координат, де кожна з наступних не пов’язана із попередніми, вже не є задовільною. Втім досить легко можна розробити ускладнену процедуру генерації випадкових маршрутних даних, причому параметри та зовнішній вигляд яких дуже близькі до реальних рухів комбайну по полю. Реалізація, як і раніше, створена у системі Mathcad і являє собою функцію користувача, що названо GenerateWay(), і яка приймає 5 вхідних параметрів:

- широта центральної точки поля;

- довгота точки поля, що може вважатися центральною;

- допустиме відхилення рухомого об’єкту від центральної точки поля у градусах (коли воно досягається, комбайн різко міняє напрямок руху назад до центру);

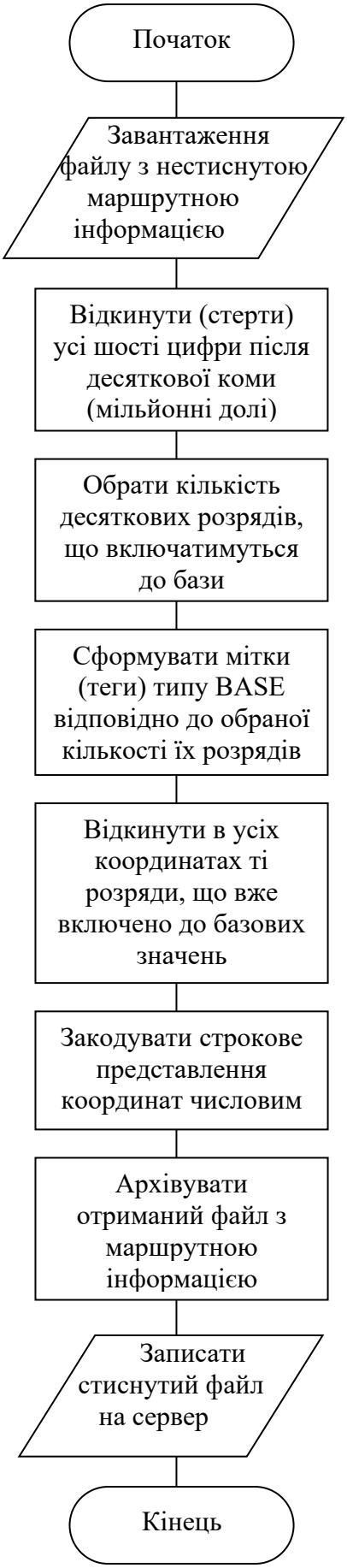


Рис. 2.3. Схема алгоритму стиснення маршрутної інформації.

- модуль швидкості комбайну у градусах за секунду;

- напрямок швидкості комбайну, що задається єдиним (полярним) кутом α;

- кількість точок для генерації.

Код розробленої функції можна побачити на рис. 2.4.

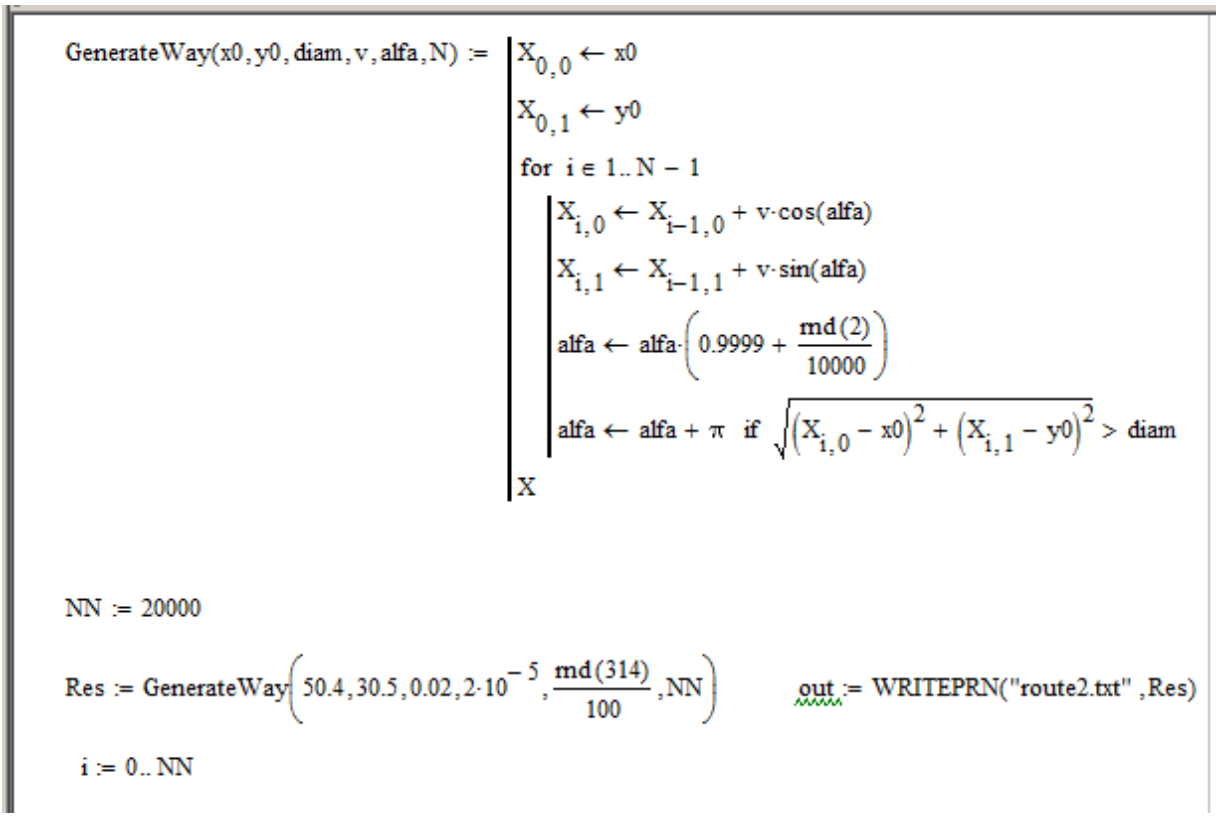


Рис. 2.4. Фрагмент вікна середовища Mathcad при моделюванні маршрутної інформації для сільгосптехніки на полі.

Результати роботи цієї функції (при виклику GenerateWay(), що показаний у нижній частині рис. 2.4) наведено на рис. 2.5.

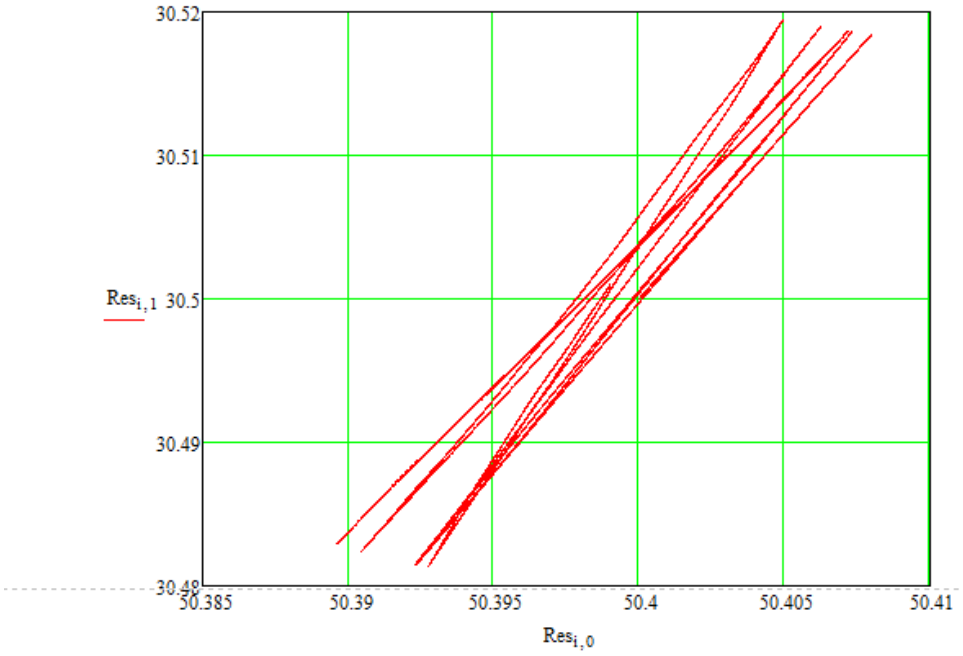


Рис. 2.5. Приклад генерації модельних даних про маршрут сільськогосподарської техніки (у межах одного поля).

Відповідний «великий» набір координат записується у файл (route2.txt) і може бути використаний для застосування методів стиснення, зокрема, відповідно алгоритму рис. 2.3.

2.4. Урахування залежності особливостей методів та моделей мінімізації обсягів інформації від швидкості руху об’єкту.

Як уже зазначалося у першому розділі, параметри маршрутної інформації в цілому залежать від величини швидкості рухомого об’єкту. Зокрема, найяскравішим вираженням цієї залежності є те, що при рівності швидкості нулю (тобто при простої техніки) координати деякий час просто повторюються. Звичайно немає ніякого сенсу зберігати дані, що повторюються багато разів підряд, і найочевиднішим підходом у цьому випадку є просте уведення у файл числа повторів кожного набору координат із одночасним виключенням координат, що повторюються. Це можна

зробити за допомогою спеціального тегу або мітки (який використовувати на додачу до BASE).

Отже, окрім введення тегів типу BASE (B_X, B_Y) у файл слід ввести також тег типу REP (від REPetition – повторення), який означає повтор останніх координат N разів, що є надзвичайно ефективним та вкрай необхідним способом стиснення маршрутних даних при банальному простої техніки на узбіччі.

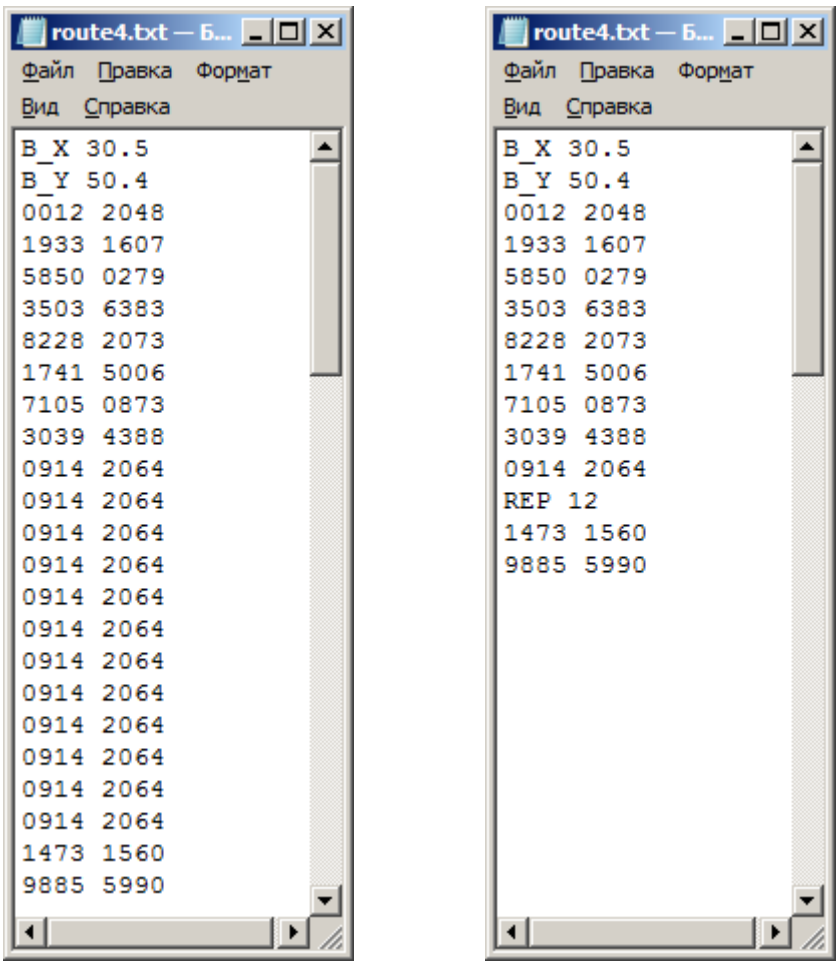
Відмітимо, що реалізовувати розділення тегу REP на дві частини окремо (типу R_X, R_Y) не має сенсу, так як це не зробить систему більш гнучкою та ефективною: якщо об'єкт не рухається, то незмінні одночасно обидві його координати, а якщо він переміщується, то мінятися будуть також обидві координати, хоча можливо в одній із них зміни будуть тільки у наймолодших розрядах координат (рух **строго** уздовж меридіану чи паралелі надзвичайно малоймовірний, тому цей випадок як реальний у практиці не розглядаємо).

При такому підході файл з координатами буде перетворений за прикладом, наведеним на рис. 2.6. Очевидно, що при тривалих простоях, або численних короткочасних зупинках даний підхід може надати значної економії обсягу місця, необхідного для збереження маршрутної інформації.

При введенні описаного тегу REP його слід впровадити у алгоритм рис. 2.3 та послідовність операцій, що йому відповідають. Відповідно, отримаємо нову схему алгоритму (рис. 2.7) та кінцеву послідовність дій при стисненні маршрутної інформації:

1) відкидання мільйонних часток (шоста цифра після десяткової коми) в *усіх* наявних координатах;

2) вибір кількості десяткових розрядів (варіант 2 або 4 із табл. 2.2), що включатимуться до бази, та, відповідно, інших, що залишатимуться у кожній координаті (залежно від характеру маршрутної інформації; для випадку руху сільськогосподарської техніки по полю оптимальним є варіант 4 із табл. 2.2);



а) *б*)

Рис. 2.6. Фрагмент файлу з маршрутною інформацією: *а* – до стиснення; *б* – після застосування тегу REP.

3) формування міток типу BASE, в яких задається базова (незмінна в рамках великого блоку даних) початкова частина кожної координати, та відкидання цієї базової частини у межах одного блоку в усіх наявних у блоці координатах;

4) формування міток типу REP та відкидання координат, що повторюються, якщо хоча б дві, або більше підряд розташованих точки мають однакові координати (при простої або зупинці техніки);

5) виконання кодування кожного чотиризначного числа, що відповідає одній із координат, однією числовою змінною типу short int – за умови, що у кроці 2 було обрано варіант №2 із табл. 2.2;

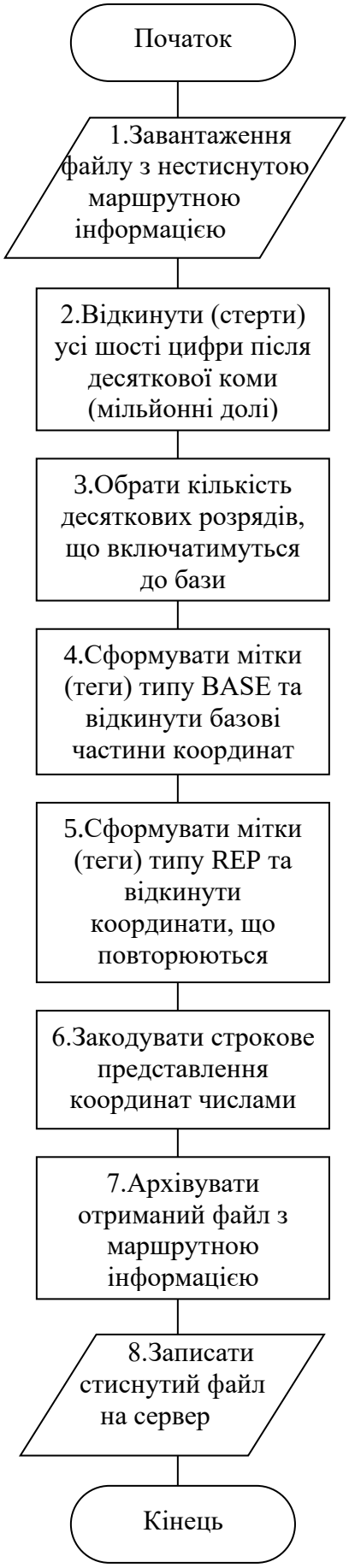


Рис. 2.7. Кінцевий варіант алгоритму стиснення маршрутної інформації (з урахуванням повторів окремих координат).

6) виконання кодування кожного двохзначного числа, що відповідає одній із координат, однією числовою змінною типу char/byte – за умови, що у кроці 2 було обрано варіант №4 із табл. 2.2;

7) архівування отриманого файлу з координатами за допомогою архіватора.

Таким чином, у даному підрозділі розглянуто варіант впровадження в систему залежності параметрів стиснення маршрутної інформації від швидкості об’єкта на прикладі ситуації зупинки або простою, коли координати, що повторюються, відкидаються, а залишається лише перше їх входження із указанням кількості повторень (запропоновано спеціальний тег REP, або коротко R).

2.5. Висновки по розділу.

Таким чином, у даному розділі запропоновано методи для стиснення маршрутної інформації за наступними принципами:

- відкидання найменших (мільйонних) долей градусу, що відповідають переміщенням на поверхні землі близьким до 0,1 м (згідно (1.7)) – через їх непотрібність у системі, що розглядається;
- розбиття кожної координати на базову частину, що прописується один раз для порівняно великих наборів підряд розташованих координат за допомогою спеціального тегу типу BASE (запропоновано окремо розглядати базу для довготи та широти: B_X та B_Y), та на другу частину – значущі молодші розряди кожної координати, що не включені до бази;
- виключення координат, що повторюються декілька разів підряд (така ситуація реалізується, якщо об’єкт стоїть на одному місці), шляхом впровадження спеціального тегу REP (або коротко R) із указанням кількості повторів передуючої йому координати;
- представлення строкових виразів (якими початково є координати точок та їхні частини-складові) у числовій формі, тобто перехід від символьного кодування до двійкового;

- використання традиційних програм архіваторів перед кінцевим вміщенням на файловий архівний сервер.

Усі ці дії роблять можливим ефективне стиснення маршрутної інформації перед вміщенням її у централізований файловий архів. Наступним кроком у роботі має стати реалізація кінцевого рішення у вигляді завершеного програмного продукту, чому і присвячений наступний розділ.

РОЗДІЛ 3. ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ СИСТЕМИ МІНІМІЗАЦІЇ МАРШРУТНИХ ДАНИХ ДЛЯ СИСТЕМ НАКОПИЧЕННЯ В УМОВАХ МОБІЛЬНОСТІ (ПЛАТФОРМИ SOC)

3.1. Обґрунтування вибору засобів розробки.

У попередньому розділі виконано розробку алгоритму ефективної мінімізації маршрутних даних, що підлягає подальшому впровадженню у програмній реалізації.

Перед виконанням будь-якої програмної реалізації слід визначитися із кількома концептуальними питаннями, пов’язаними із нею, а саме:

- яку технологію програмування найкраще застосувати для реалізації даного алгоритму при даному комплексі умов;
- яку мову програмування, що підтримує обрану технологію програмування, доцільніше всього застосувати для програмної реалізації у наявних умовах;
- яке середовище розробки (або комплекс простих окремих засобів розробки) краще всього застосувати для даної програмної реалізації.

Тільки отримавши обґрунтовані відповіді на ці запитання, можна переходити безпосередньо до етапу програмування.

3.1.1. Вибір технології розробки.

Отже, першочерговим питанням, яке постає перед розробниками практично будь-якого програмного забезпечення, є вибір моделі або технології його розробки. Такими, що реально широко використовуються на сьогоднішній день у виробничій практиці, є технології структурного (процедурного) та об’єктно-орієнтованого програмування. Кожна з них має свої особливості, переваги і недоліки, які розглянемо докладніше.

Структурне, або як його ще називають, процедурне програмування засноване на використанні окремих структурних блоків - в першу чергу, підпрограм (процедур і функцій).

Історично перші комп’ютерні програми були відносно простими і мали пакетний режим роботи: отримуючи на вхід якусь інформацію (можливо,

навіть на перфокарті) вони виконували певний обсяг операцій з обробки цих даних і видавали результат. При цьому існувала сувора функціональна залежність виходу від входу – рис. 3.1.

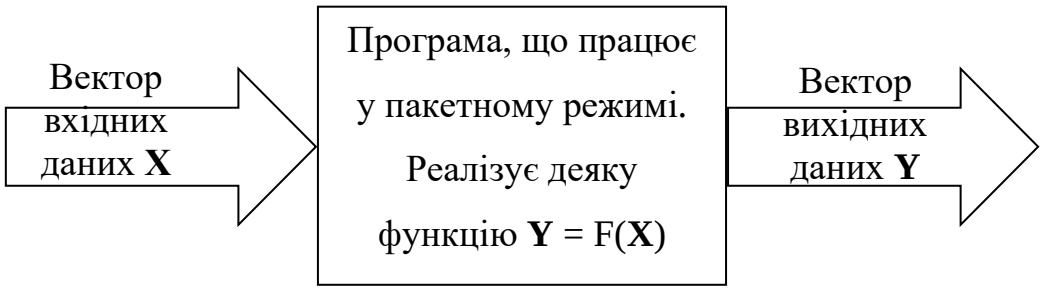


Рис. 3.1. Схема роботи пакетної програми.

Незважаючи на відсутність явного зв’язку функціональності і структури, пакетні програми в основному мали просту лінійну послідовність виконання. Тобто в них були практично відсутні будь-які підпрограми, принаймні, використання підпрограм не було важливим елементом самої методики програмування.

В міру ускладнення функціональності програмного забезпечення, змінювалася і його внутрішня структура: поступово розвинулася інтерактивна модель взаємодії користувача і програми – рис. 3.2. Програми стали запитувати інформацію і активно реагувати на дії людини. Ускладнення функціональності привело до відповідного ускладнення програмного коду, який, в першу чергу, став досить обширним у розмірах. Обширним для того, щоби середньостатистична людина-програміст без всяких спеціальних хитрувань швидко і легко розібралася з незнайомим кодом. Розміщувати код у простій лінійній послідовності без виділення великих блоків стало незручно, в першу чергу, для розуміння цього коду.

Тут слід зазначити психологічні особливості сприйняття людиною складних «великих» завдань. Неструктуроване «велике» завдання (наприклад, написання дипломної роботи) зазвичай викликає певний психологічний ступор і, як результат, повну неможливість поступово розібратися з ним. Людині зручно розбити проблему на не надто велику

(зазвичай до десятка, а краще 3-4) кількість завдань (наприклад, розділів у дипломній роботі), не замислюючись про реалізацію кожного з них. Коли є ясність і розуміння проблеми на найвищому рівні абстракції, слід приступати до деталізації підзадач, кожну з яких слід розбити на окремі «підпідзадачі» тобто підзадачі нижчого рівня, більш дрібні. Уже після такого розбиття слід аналізувати всі перераховані підзадачі. На певному етапі зупиняються і виконують не розбиття чергової підзадачі на більш дрібні, а безпосередню її реалізацію в програмних кодах.

Рис. 3.2. Схема роботи програми, що активно взаємодіє з користувачем.

Отже, можна сказати, що розвиток методик програмування відбувався паралельно з розвитком призначеного для користувача інтерфейсу: і грубо кажучи, розвиненому інтерфейсу командного рядка відповідає парадигма структурного програмування.

Говорячи більш формалізовано, структурне програмування має на увазі побудову програми відповідно до трьох основних принципів: слідування, розгалуження, повторення.

Слідування означає, що оператори і блоки програмного коду слідують і виконуються один за іншим. Розгалуження реалізується різними умовними операторами типу if, і дозволяє вибирати один з декількох подальших варіантів виконання програми. Повторення зазвичай відносять на рахунок

циклів (багаторазових повторів однієї і тієї ж ділянки коду, що йдуть підряд), хоча цей же принцип можна віднести і до підпрограм.

Взагалі ж, структурне програмування іноді називають застарілою методикою програмування, на зміну якій разом з віконним інтерфейсом прийшло об'єктно-орієнтоване програмування (деякі сучасні мови програмування загального призначення навіть не дозволяють створити структурну програму, тільки об'єктно-орієнтовану – як, наприклад, Visual C# чи Java). Проте, при створенні невеликих програм (наприклад, до 10000 рядків коду і без передбачуваного розширення) застосування цієї методики програмування однозначно більш виправдано і відповідний код набагато краще сприймається, ніж його об'єктно-орієнтований варіант.

Суть же методології об'єктно-орієнтованого програмування полягає в тому, що система розглядається, як сукупність окремих сутностей - об'єктів, які мають набір якихось своїх внутрішніх параметрів - властивостей, а також можуть взаємодіяти між собою за допомогою деяких дій - викликів методів (або трохи більше непрямим чином - шляхом надсилання повідомлень, оброблюваних методами об'єктів; для цього необхідна присутність активної сутності, яка роздає повідомлення адресатам, як, наприклад, менеджер вікон в ОС Windows).

Якщо говорити про програмний код, то для того, щоб оперувати об'єктом, його спочатку потрібно створити. Об'єкти створюються як змінні, у яких типом виступає клас об'єкта. Клас - це просто опис, які властивості можуть мати об'єкти такого типу (тобто яку інформацію вони можуть зберігати), і які у них є методи (тобто які дії вони можуть виконувати). Об'єкт - це набір значень, чому саме рівні властивості даного об'єкта (свої методи кожен об'єкт отримує від свого класу, тобто методи однакові у всіх об'єктів, що належать даному класу).

Для чого потрібен цей специфічний підхід, адже самі по собі об'єкти не додають нічого корисного (навпаки, введення об'єктів ускладнює програму, вносить в неї нові сутності)? Виявляється, реалізовуючи всі сутності,

необхідні, згідно з алгоритмом, для роботи програми, у вигляді класів і об’єктів, ми спрощуємо її розуміння для самих себе. Саме тому ОО-підхід рекомендується до застосування для великих проектів (більше десятків тисяч рядків коду), коли утримувати «в голові» всю систему цілком стає важко. Можна сказати, що розбиття програми на об’єкти і проектування їх класів наближає розуміння предметної області до звичного людського образу мислення (в разі «великих» проектів). Людина мислить класами, об’єктами і зв’язками між ними.

Порівняльна складність проектів, що мають однакову функціональність, але побудованих по-різному (згідно структурному і об’єктно-орієнтованого підходів до програмування), як функція їх обсягу показана на рис. 3.3. З графіка рис. 3.3 слід, що, якщо потрібно реалізувати продукт з невеликою функціональністю (тобто кількість рядків коду, що її реалізують буде очевидно невеликою, порядку кількох тисяч рядків), то це краще робити без застосування класів, так як вони будуть тільки ускладнювати всю справу. Якщо ж програма має більш-менш значну функціональність, а значить, реалізується хоча б кількома тисячами рядків коду, то вже є сенс замислюватися про застосування об’єктно-орієнтованого підходу. Однозначно будуватися за принципами ООП повинні програми, які мають 10000 рядків коду і більш.

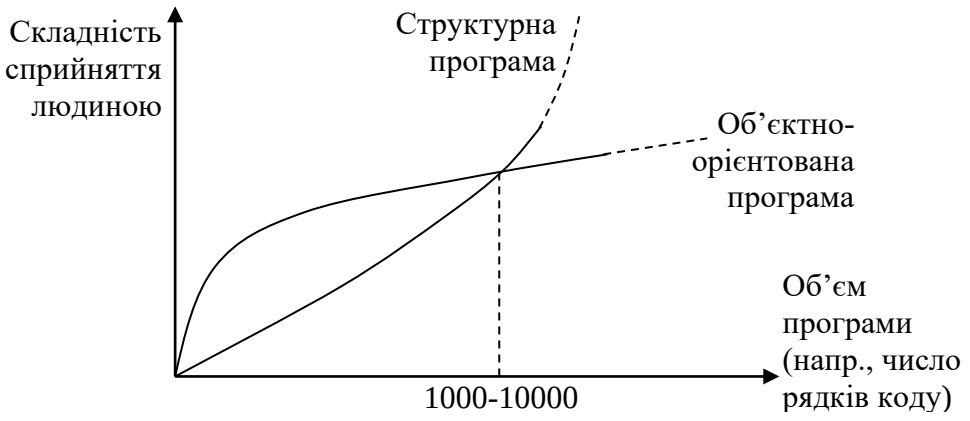


Рис. 3.3. Порівняльна складність висхідного тексту двох програм, що мають однакову функціональність, але реалізованих по-різному: згідно об’єктно-орієнтованому та структурному підходам.

Відзначимо, що часто крім розглянутих міркувань, також на вибір методики програмування впливають інші чинники, наприклад, можливість майбутнього розширення функціональності, створення якомога більш зрозумілого коду (для роботи над проектом цілої команди, а не одного програміста), або просто побажання замовника застосувати найбільш сучасний підхід до програмування (чи навпаки, дозвіл на використання вікових, перевірених часом технологій).

Крім розбиття (декомпозиції) всієї предметної області на об'єкти (класи) і співвідношення між ними, також ОО-підхід має на увазі дотримання трьох основних його принципів: інкапсуляція, наслідування, поліморфізм.

Під інкапсуляцією мається на увазі об'єднання даних (значення властивостей класу у деякого конкретного об'єкта) та засобів їх обробки (методи класу). Це знову ж таки зручно психологічно, так як дозволяє реалізовувати окремі завершені сутності - класи, які самі обробляють свої дані. Звернення до об'єктів цих класів відбувається за допомогою методів, що утворюють інтерфейс класу.

Наслідування дуже корисно, тому що дозволяє сильно скоротити обсяги повторюваного коду (до чого потрібно завжди прагнути при розробці будь-якого програмного забезпечення). Згідно з цим принципом виділяється клас, який має загальний набір властивостей і методів для декількох більш розширених класів. Цей клас оголошується батьком, базовим класом для декількох похідних від нього (нащадків, спадкоємців). Всі класи-нащадки успадковують від базового всі його властивості та методи, але до цього ще мають свої власні оригінальні властивості і / або методи.

Наприклад, клас Студент є похідним від класу Людина, тому що кожна людина має властивість Ім'я, Прізвище, метод Відпочити(). Однак у Студента є свої специфічні властивості і / або методи, які як раз і відрізняють його від просто Людини: СереднійБал, НомерЗаліковки, ЗдатиЕкзамен(), і т.д.

При наслідуванні іноді методи батьківського і похідного класу мають однакове призначення, але реалізовуються по-різному. Такі методи називаються перевантаженими. Наприклад, метод Відпочити() у класу Людина реалізується як відпочинок на дивані, а у класу Студент - як похід в клуб. При цьому ще раз підкреслимо, що призначення методу в обох випадках одне і той саме.

Поліморфізм є можливістю деякої функції приймати об’єкти як батьківського, так і похідних класів, і вміти викликати перевантажені методи саме того класу, об’єкт якого був переданий в функцію. Слід сказати, що це досить специфічна можливість і в загальному багато програмістів використовують ОО-підхід і без звернення до поліморфізму.

Нехай, наприклад, у програмі є функція ПровестиВихідні(), припустимо яка не належить якомусь класу (хоча це не принципово). Нехай аргументом цієї функції є об’єкт класу Людина. Тоді в неї можна передавати об’єкти всіх похідних від Людини класів: Студент, Службовець, Пенсіонер, і т.д., тому що всі вони є Людиною (спадкоємці цього класу). Ясно, що ця функція повинна включати різні дії: ПрибратиКвартиру(), ПітиНаРинок(), і в тому числі Відпочити(). Так ось поліморфізм дозволяє всередині цієї функції просто вказати назву методу Відпочити(), не вказуючи якого саме класу він повинен бути викликаний, а вже в процесі виконання програми, якщо в функцію переданий об’єкт класу Студент, то викликається саме його метод Відпочити(), а якщо переданий об’єкт класу Пенсіонер, то автоматично викликається саме його метод Відпочити(), і т.д. Кажуть, що функція ПровестиВихідні() - поліморфна, і вона є такою завдяки тому, що реалізує принцип поліморфізму.

Важливими поняттями в ООП також є: статичні члени класу, абстрактні методи і класи, дружба функцій і класів, і т.д.

Грунтуючись на перерахованих особливостях двох існуючих методів програмування, вибираємо структурний підхід як більш простий, що відповідає невеликим (не промисловим) масштабам проектованого ПЗ, а

також вимогам до складності (програма не може бути складною через однотипність операцій, виконуваних нею: в основному це операції з рядками).

3.1.2. Вибір мов програмування.

Після вибору технології програмування наступним кроком є вибір мови програмування (або комплексу мов – за умови, наприклад, написання програмного продукту у вигляді розподіленого клієнт-серверного додатку, а також при деяких інших умовах).

Найбільш укрупнено за типом платформи кінцевого програмного продукту усі мови програмування можна розподілити на засоби для розробки настільних додатків, засоби веб-розробки та мобільної розробки. Мобільні технології в даній роботі взагалі не можуть стати в нагоді, оскільки розроблюваний програмний продукт не потребує мобільності, а обчислювальні ресурси мобільних систем в рази менше, ніж у настільних, чи тим більше, клієнт-серверних. Що ж стосується альтернативи «веб-додаток» чи «настільне програмне забезпечення», перший варіант видається зручнішим через наступні причини:

- є повністю кросплатформенним (якісні веб-додатки однаково функціонально працюють у будь-яких сучасних веб-браузерах, незалежно від того, на якій операційній системі вони запущені);

- дозволяє використовувати обчислювальні ресурси серверу, які завжди значно потужніші, ніж у настільних ПК;

- можуть мати досить швидкий та зручний інтерфейс, за умови використання у них сучасних веб-технологій (типу AJAX, кеширування, локальної доробки даних, предзавантаження сторінок, і т.д.).

Таким чином, обираємо веб-розробку та проаналізуємо засоби, що можуть при такому випадку бути використані.

Сучасні засоби для розробки Інтернет-ресурсів є настільки обширними, що розібрати їх усі в рамках дипломної роботи абсолютно не уявляється можливим. Тому в процесі вибору будемо орієнтуватися як на об’єктивні

фактори (такі, як, наприклад, підтримка сучасних технологій програмування), так і суб'єктивні, але такі, що набули широкого впливу (як-то схильність окремих людей до певних мов, чи, навіть, стилів програмування, що у результаті спричинює популярність відповідних засобів розробки на масовому рівні).

Таким чином, розглянемо найпоширеніші засоби розробки, що є популярними на даний момент у відповідній галузі.

В цілому увесь процес веб-розробки традиційно ділять на дві компоненти: front-end та back-end – рис. 3.4.

3.1.2.1. Засоби Front-end розробки.

Спрощено кажучи, під «передньою» частиною – Front-end – у програмуванні мають на увазі те, що бачить користувач. Для настільних систем під цим мають на увазі інтерфейс, а під back-end'ом мають на увазі логіку роботи програми, яка теоретично (зокрема, так часто роблять у Linux-системах) взагалі може бути реалізована окремою програмою із текстовим консольним інтерфейсом. Задачею фронт-енду при цьому є зручний для широкого кола користувачів збір усіх вхідних даних, необхідних для виконання змістовних операцій консольною утилітою back-end'ом. У багатьох же випадках фронт та бек поєднані разом в рамках одного програмного забезпечення. Такою є ситуація для настільних систем.

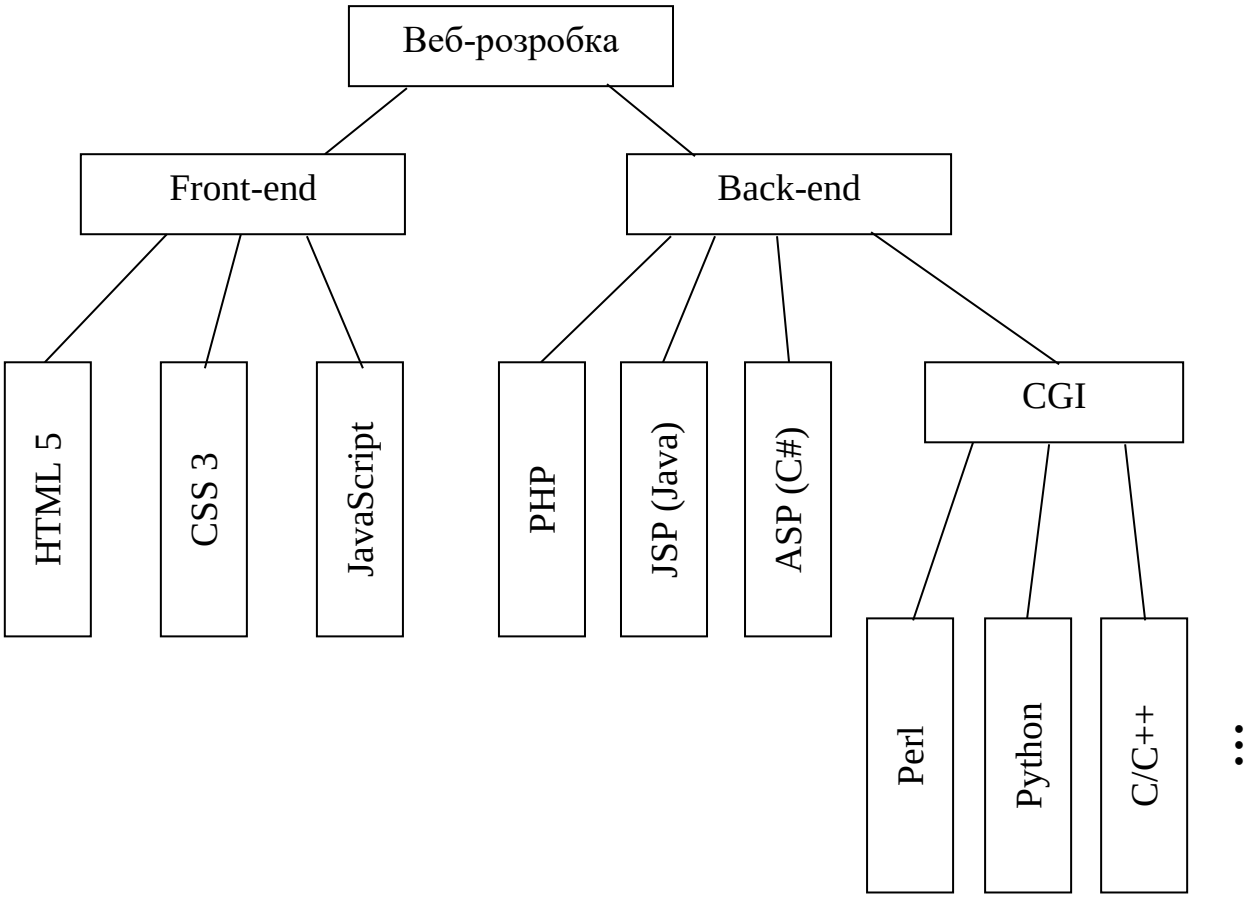


Рис. 3.4. Найпопулярніші засоби розробки сучасних Інтернет-ресурсів.

Якщо конкретніше говорити про веб-програмування, то тут ситуація є максимально поляризованою (ще більше, ніж у описаному вище випадку написання фронт-енду одним програмістом для бек-енд-утиліти, створеної іншим розробником), оскільки користувач працює і «бачить» систему через браузер на одній віддаленій машині (на комп'ютері, що називається клієнтом), а логіка додатку та практично уся супутня інформація розміщується на зовсім іншому комп'ютері-сервері.

Таким чином, усі програмні коди та висхідні тексти, що виконуються у браузері, тобто на стороні клієнта, відносяться до front-end частини. З іншого боку, все, що виконується на сервері, відноситься до back-end складової.

Якщо глибоко не вдаватися у детальний аналіз, то коротко можна сказати, що набір інструментів для front-end розробки значно менший, ніж для беку. Серед них можна виділити:

- HTML – п’ятої версії (HTML5), найсучасніший стандарт мови гіпертекстової розмітки (HyperText Markup Language), де традиційно під гіпертекстом мають на увазі текст, оснащений гіперпосиланнями на інші частини документів та, можливо, супроводжуваний картинками, звуками, відео, і т.п.;

- CSS – третьої версії (CSS3) – правила запису стилів різноманітних елементів веб-сторінки, або, якщо говорити точно – каскадні таблиці стилів;

- JavaScript – найпотужніший інструмент front-end розробки, що дозволяє вивести html-сторінки на новий рівень практично повноцінних додатків, на зразок настільних, що активно реагують на дії користувача, передають і отримують інформацію, від нього.

Ці технології є безальтернативними, тому розглянемо їх докладніше.

1) Hyper Text Markup Language (HTML) - мова розмітки гіпертексту - призначена для написання усіх документів в мережі World Wide Web - WWW.

HTML документ - це текстовий файл, який має спеціальні мітки, що називаються тегами, які при затребуванні клієнтом передаються із комп’ютера-серверу на браузер клієнта і використовуються ним для відображення вмісту файлу па екрані комп'ютера.

За допомогою цих міток можна виділяти заголовки документа, змінювати колір, розмір і написання літер, вставляти графічні зображення і таблиці. Але основною перевагою гіпертексту перед звичайним текстом є можливість додавання до вмісту документа гіперпосилань - спеціальних конструкцій мови HTML, які дозволяють клацанням миші перейти до перегляду іншого документа. Таким чином «гіпертекст» означає «надтекст» - розвинення ідеї текстового документу, але із більш широкими можливостями.

Сама по собі мова HTML є різновидом більш загальної мови XML (eXtensible Markup Language), причому з одного боку звужує її властивості, а з іншого – навпаки деталізує. Так, у мові XML допустимим є використання

будь-яких тегів, тобто з довільними іменами, в той час, як назви тегів мови HTML є цілком фіксованими, а інші слова окрім стандартних, використовувати не можна. З іншого боку, той вміст веб-сторінки, що розміщений біля конкретних тегів (точніше – між парою однакових тегів), набуває визначених стандартом властивостей (на відміну від тексту, що розміщений поруч із тегами XML, які самі по собі практично нічого не означають, а використовуються лише при наявності певних домовленостей про відображення й обробку того, чи іншого тегу) відображується браузером відповідно до суті цих тегів: якщо тегом є <B>, то текст зображуватиметься жирним, <i> - курсивом, і т.д.

HTML-документ має дві складові: власне текстова інформація, тобто дані, що складають вміст документа, і теги - спеціальні конструкції мови HTML, які використовуються для розмітки документа і керують його відображенням. Теги мови HTML визначають, в якому вигляді буде представлений текст, які його компоненти будуть виконувати роль гіпертекстових посилань, які графічні або мультимедійні об’єкти повинні бути включені в документ.

Графічна та звукова інформація, що включається в HTML-документ, зберігається в окремих файлах. Програми перегляду HTML-документів називають браузерами, і вони інтерпретують теги розмітки і оперують текстом і графікою, розміщуючи їх на екрані відповідним чином. Для файлів, що містять HTML-документи використовується розширення .htm або .html.

У переважній більшості випадків теги використовуються парами. Пара складається з тега, що відкриває <ім’я_тега> і тега, що закриває </ім’я_тега>. Дія будь-якого парного тега починається з того місця, де зустрівся відкриваючий тег, і закінчується при зустрічі відповідного закриваючого тега. Часто пару, що складається з відкриваючого і закриваючого тегів, називають контейнером, а частину тексту між відкриваючим і закриваючим тегом - елементом.

Послідовність символів, яка утворює текст, може складатися з пробілів, табуляцій, символів переходу на новий рядок, символів повернення каретки, букв кириличного та латинського написань, знаків пунктуації, цифр, і спеціальних символів (наприклад #, +, $, @), за винятком наступних чотирьох символів, що мають в HTML спеціальний сенс: < (менше), > (більше), & (амперсанд) і «(лапки)». Якщо необхідно включити в текст будь-який із цих спецсимволів, то слід закодувати його особливою послідовністю символів:

```
<     -     &lt
>     -     &gt
&     -     &amp
“     -     &quot
```

Найпершим із тегів HTML розміщується однойменний тег <html>. Він завжди відкриває документ, так само, як тег </html> повинен неодмінно стояти в останньому його рядку. Ці теги позначають, що рядки, які між них знаходяться, представляють собою єдиний гіпертекстовий документ. Без цих тегів браузер або інша програма перегляду не в змозі ідентифікувати формат документа і правильно його інтерпретувати.

Далі, якщо говорити укрупнено, HTML-документ має дві частини: заголовок (head) і тіло (body), розташованих в наступному порядку:

```
<Html>
<Head> Заголовок документа </ head>
<Body> Тіло документа </ body>
</ Html>
```

Найчастіше в заголовок документа включають парний тег <title> ... </title>, що визначає назву документа. Багато програм перегляду використовують його як заголовок вікна, в якому виводиться документ. Програми, що індексують документи в мережі Інтернет, використовують назву для ідентифікації сторінки. Гарна назва повинна бути досить довгою для того, щоб можна було коректно вказати відповідну сторінку, і в той же час воно має міститися в заголовку вікна. Назва документа вписується між відкриваючим і закриваючим тегами title.

Тіло документа є обов'язковим елементом, так як в ньому розташовується весь матеріал веб-сторінки. Тіло документа розміщується між тегами <body> і </body>. Все, що розміщено між цими тегами, інтерпретується браузером відповідно до правил мови HTML дозволяють коректно відображати сторінку на екрані монітора.

Текст в HTML розділяється на абзаци за допомогою тега . Він розміщується на початку кожного абзацу, і програма перегляду, зустрічаючи його, відокремлює абзаци один від одного порожнім рядком. Використання закриваючого тега є необов'язковим.

Якщо потрібно «розірвати» текст, перенісши його залишок на новий рядок, при цьому, не виділяючи нового абзацу, використовується тег розриву рядка
. Він змушує програму перегляду виводити символи, що стоять після нього з нового рядка. На відміну від тега абзацу, тег
 не додає порожній рядок. У цього тега немає парного закриваючого тега.

Мова HTML підтримує логічне н фізичне форматування вмісту документа. Логічне форматування вказує на призначення даного фрагмента тексту, а фізичне форматування задає його зовнішній вигляд.

При використанні логічного форматування тексту браузером виділяються різні частини тексту відповідно до структури документа. Щоб відобразити назву, використовується один з тегів заголовка. Заголовки в типовому документі поділяються за рівнями. Мова HTML дозволяє задати шість рівнів заголовків: h1 (заголовок першого рівня), h2, h3, h4, h5 і h6. Тема першого рівня має зазвичай більший розмір і насиченість в порівнянні з заголовком другого рівня. Приклад використання тегів заголовків:

```
<Hl> l. Назва розділу </hl>
<H2> l.l. Назва підрозділу </h2>
```

Теги фізичного форматування безпосередньо задають вид тексту на екрані браузера, наприклад пара   виділяє текст напівжирним шрифтом, <u> </u> задає підкреслення тексту,   керує шрифтом тексту. Саме ці елементи вважаються застарілими, і замість них у

специфікації HTML5 рекомендується користуватися стилями, що розглянемо нижче.

Тег <img> вставляє зображення в документ, причому так, якби воно було просто одним великим символом. Закриваючий тег для зображення не потрібний. Приклад застосування тега:

```
<Img src = "picture.gif">
```

Для створення гіпертекстового посилання використовується пара тегів <a> ... . Фрагмент тексту, зображення або будь-який інший об'єкт, розташований між цими тегами, відображається у вікні браузера як гіпертекстове посилання. Активація такого об’єкта за допомогою щигля мишею призводить до завантаження у вікно браузера нового документа або до відображення іншої частини поточної Web-сторінки. Гіпертекстове посилання формується за допомогою формули:

```
<A href = "document.html"> посилання на документ </a>
```

Href тут є обов'язковим атрибутом, значення якого і є URL-адресою запитуваного ресурсу. Лапки в завданні значення атрибута href не обов'язкові (як і при завданні значень і для інших атрибутів будь-яких тегів, але використання лапок – подвійних, чи одинарних, є вкрай бажаним). Якщо задається посилання на документ на іншому сервері, то вид гіперпосилання такий:

```
<A href = "http://www.school.dn.ua/11.jpg">Фотографія 11-А
</ a>
```

Підсумовуючи можливості мови HTML, можна сказати, що за допомогою різних тегів можна малювати таблиці, форматувати текст, вставляти в документ зображення, відео-, звукові файли та інше. В процесі свого розвитку все більша увага приділялася тегу <style> та пов’язаними з ним каскадними таблицями стилів.

Каскадні таблиці стилів (Cascading Style Sheets, CSS) - це мова, яка містить набір властивостей для визначення зовнішнього вигляду документа. Специфікація CSS (CSS3 – на даний момент) визначає властивості і описову мову для встановлення зв'язку властивостей з елементами в документі.

Розуміння таблиць стилів необхідно для додавання динамічного стилю сторінки. Під динамічним стилем (dynamic style) тут маємо на увазі модифікацією таблиці стилів, пов'язану з документом за допомогою сценарію.

На вузлі консорціуму W3C (www.w3.org) можна знайти останню інформацію про нововведення і елементах, підтримуваних таблицями стилів.

Таблиці стилів представляють собою абстракцію, в якій стиль документа визначається окремо від змісту або структури. Існує три методи додавання таблиць стилів в документ, доступних для Web-майстра - в цілому, з підвищенням рівня складності розширюються надані можливості з одночасним збільшенням ступеня абстракції. Перший метод полягає в використанні таблиці внутрішніх стилів (inline style sheet). Внутрішні стилі (inline styles) визначаються безпосередньо в елементі. Другий метод полягає в використанні таблиці глобальних стилів (global style sheet) для визначення стилю на початку документа. Третій, найбільш абстрактний і потужний метод полягає в використанні таблиці пов'язаних стилів (linked style sheet) для визначення стилю окремо в іншому документі.

Внутрішні стилі мало відрізняються від традиційного HTML. При використанні внутрішніх стилів зовнішній вигляд документа важко змінити. Перевага даного методу полягає в скороченому обсязі розмітки і в тому, що HTML може бути більш повноцінно використаний для подання вмісту презентації. Використання таблиці глобальних стилів дозволяє більш ефективно відокремити представлення від вмісту, а також дає можливість швидкої і незалежної зміни стилю і обробки документа. Використання таблиці пов'язаних стилів дає більше переваг, представляючи вміст у вигляді набору сторінок або визначаючи цілий Web-вузол за допомогою одного файлу.

Термін cascading (каскадні) в назві CSS вказує на можливість злиття різних таблиць стилів для створення єдиного визначення стилю для елемента або для цілого документа. Це дозволяє проводити передбачуване злиття

таблиці стилів Web-вузла з таблицею стилів документа і навіть з внутрішнім стилем.

Отже, внутрішній стиль (inline style) є по суті таблицею стилів для одиночного екземпляру елемента і його визначено безпосередньо у тегу елемента. Таблиця внутрішніх стилів визначається з використанням атрибута STYLE, а дані для атрибута визначаються за допомогою мови таблиці стилів. Нижче наведено код HTML, який збільшує шрифт відображення вмісту параграфа і вирівнює параграф по центру на жовтому фоні:

```
    <P STYLE = "font-size: 120%; text-align: center; background: yellow">
    Створює жовтий вирівняний по центру параграф з великим розміром шрифту.
    </ P>
```

Внутрішні стилі допомагають при вивченні мови CSS або за необхідності швидко змінити одиночний екземпляр елемента. Однак, внутрішні стилі не відповідають ідеології структурованого документа і погано працюють при необхідності зміни зовнішнього вигляду ряду елементів в документі, коли презентація та вміст розділені в повному обсязі. Для відділення стилю документа від його структури таблиця стилів повинна бути визначена в заголовку документа або як окремий файл, який пов'язаний з документом.

Наступною, більш ідеологічно вірною можливістю, є використання окремого тегу <STYLE> для завдання таблиць глобальних стилів, який зазвичай розміщується в заголовку документа. Розміщення усіх стилів документа в одному місці спрощує зміну режиму відтворення документа. Наведений нижче приклад таблиці стилів визначає зовнішній вигляд усіх параграфів в документі. Для зміни режиму відтворення параграфів потрібно змінити тільки елемент STYLE. Якби використовувалися внутрішні стилі, то довелося б міняти кожен параграф в документі окремо.

```
<HTML>
  <HEAD>
     <STYLE TYPE = "text / css">
```

```
            P {font-size: 120%; text-align: center; background:
yellow}
         </ STYLE>
      </ HEAD>
      <BODY>
         <P> Все параграфи тепер більше і вирівняні по центру
на жовтому
            тлі. </ P>
      </ BODY>
   </ HTML>
```

Для зв'язку стилю з певним елементом використовується селектор (selector). У наведеному вище прикладі був створений простий селектор, який пов'язаний зі стилем у всіх параграфах. Можуть бути також визначені більш функціональні контекстуальні селектори, що описуються нижче у цьому розділі.

Ще одним із трьох способів завдання стилів є введення таблиці пов'язаних стилів (linked style sheet), яка знаходиться у зовнішньому файлі. Перевага використання таблиці пов'язаних стилів полягає в тому, що всі правила і стилі можуть бути визначені і вміщені в одному файлі, який може бути спільно використаний багатьма сторінками або навіть цілим Web-вузлом. При використанні таблиці пов'язаних стилів обробка всіх параграфів цілого Web-вузла може бути змінена в одному документі. Таблиця пов'язаних стилів може також підвищити продуктивність, оскільки вона кеширується локально на диску клієнта, окремо від документа, так що кожен документ має менший розмір, а інформацію про стилі буде потрібно завантажити тільки один раз.

Для визначення таблиці пов'язаних стилів у заголовку документа розміщується тег <LINK>:

```
    <HTML>
      <HEAD>
         <LINK REL = "stylesheet" TYPE = "text / css" HREF =
"fancy.css">
      </ HEAD>
      <BODY>
         <P>  This  document  uses  the  styles  specified  in
fancy.css. </ P>
      </ BODY>
   </ HTML>
```

Атрибут REL визначає, що пов'язаний файл є таблицею стилів, а атрибут TYPE визначає тип MIME таблиці стилів. Атрибут HREF є покажчиком URL, що вказує на зовнішню таблицю стилів. Таблиця пов'язаних стилів повинна містити тільки контекстуальні правила і визначення стилю і не може включати код HTML.

Для створення таблиці стилів всередині документа використовується той же синтаксис, який використовувався при створенні таблиці пов'язаних стилів. Мова CSS складається з селекторів і правил подання (presentation rules). Селектори (selectors) визначають елементи, які пов'язані з певним правилом, а правила подання встановлюють методи обробки даних елементів.

CSS містить два типи селекторів: прості і контекстуальні. Простий селектор пов'язує елемент на основі його атрибутів або типу незалежно від контекстуального положення усередині інших елементів. Контекстуальні елементи є більш потужними - вони можуть зв'язати правило з контейнерами певного елемента, наприклад, усі теги <EM> всередині тегів <P>.

У базовій формі простий селектор може бути створений для зв'язку певного елемента, класу елементів або ідентифікатора (ID) з певним стилем. Наведений нижче код демонструє ряд простих селекторів і їх правил подання:

```
      <STYLE TYPE = "text / css">
        / * Change all H1s to red. * /
        H1 {color: red}
        / * Встановлює напівжирний шрифт всіх елементів з тегом
CLASS = "Special" boldface. * /
        .special {font-weight: bold}
        / * Розміщує елемент з ідентифікатором ID = special на
жовтому тлі.*/
        #special {background: yellow}
        / * Встановлює висновок елементів H1 з тегом CLASS =
"cool" з великим інтервалом.* /
        H1.cool {letter-spacing: 2px}
     </ STYLE>
```

Селектори можуть бути перераховані через кому, що дозволяє одночасно описати кілька селекторів:

```
/ * Встановлює для всіх заголовків однакові правила. * /
H1, H2, H3, H4, H5, H6 {color: red; background: yellow}
```

Контекстуальні селектори визначають ієрархію контейнерів, з якою повинен бути пов'язаний стиль. Ієрархія контейнерів визначається порядком елементів в списку через кому. Наприклад, наведений нижче оператор визначає правило для всіх елементів EM, що знаходятцься в елементі P:

```
P EM {color: blue}
```

Кожен селектор може посилатися на теги CLASS, ID або тип елемента. Нижче наведена більш складна версія контекстуального селектора:

```
 / * Будь-який елемент з CLASS = "cool", який знаходиться всередині елемента LI з CLASS = "special" і далі перебуває всередині елемента UL, буде використовувати даний стиль. * /
UL LI.special .cool {font-weight: bolder; font-size: 120%}
```

Всі елементи контекстуального селектора є нечутливими до регістру - наприклад, .cool це те ж саме, що і .cOoL.

Псевдоклас (pseudo-class) складається з елементів одного типу, які задовольняють певному контекстуальному критерію. Наприклад, переглянуті елементи Anchor (посилання) представляють собою псевдоклас visited. Активні і не переглянуті посилання являють собою псевдокласи active і link, відповідно. Псевдоклас в таблиці стилів відділяється двокрапкою:

```
A: link {color: green}
: Link {color: green}
```

У другому прикладі опущено ім'я елемента (A), так як тільки посилання мають псевдоклас link. Псевдоклас може бути використаний так само, як клас або покажчик ID і є нечутливим до регістру.

На одні й ті ж самі елементи можуть посилатися кілька селекторів. CSS визначає послідовність каскадування (cascading order), яка використовується для вирішення проблем перекривання областей дії селекторів і правил. Послідовність каскадування об'єднує всі правила, які застосовуються до елементу, шляхом сортування на основі їх визначень. Наприклад, елемент Strong, що знаходиться в елементі H1, може мати правила подання, визначені селектором H1, селектором STRONG і контекстуальним селектором для

елементів Strong всередині елементів H1. Параметр каскадування в CSS визначає порядок об'єднання даних трьох правил. Загалом, правило для більш конкретного контекстуального селектора відкидає правило для менш конкретного селектора, а правила, подані нижче у початковій таблиці стилів або документі, мають більш високий пріоритет.

Докладно розглянувши каскадні таблиці стилів слід також подивитися основні особливості останнього, дуже потужного засобу front-end-розробки, а саме – мови програмування JavaScript. Це спеціалізована мова програмування, яка традиційно використовується для управління об’єктною моделлю документа у браузері під час перегляду сторінок мережі Інтернет (існують продукти, що перетворюють JavaScript на мову загального призначення, команди якої виконуються на сервері, наприклад, Node.JS; але такий підхід сильно протирічить початковій концепції використання цієї мови програмування, і, за умови наявності значної кількості традиційних засобів back-end-розробки перетворення на серверну мову ще й JavaScript значна кількість програмістів вважає абсолютно недоцільною, тому про цей варіант використання говорити більше не будемо).

Особливістю JavaScript є те, що його висхідні програмні коди інтерпретуються, що є досить гнучким, але повільним рішенням.

Оператори у цій, в цілому C-подібній мові, розділяються крапкою з комою. Мова чутлива до регістру, що часто випускають з виду початківці, ймовірно тому, що HTML, застосовуваний зазвичай з JavaScript спільно, не залежить від регістру (імена тегів і атрибутів HTML можна писати як малими, так і великими літерами).

Однорядковий коментар виділяється символом //, а багаторядковий - парою символів / * і * /, в чому JavaScript знову повторює C.

До даних застосовується слабкий (динамічний) контроль типів. В операторах з різнотипними даними останні автоматично приводяться до необхідного типу. Типи даних можуть бути примітивними і складеними. Примітивні типи містять прості однорідні значення, такі дані можна

передавати функціям як параметри за значенням, а не за посиланням. Складені типи містять різнорідні дані (в тому числі і складені), їх передають у функції тільки за посиланням.

Мова JavaScript об’єктно-орієнтована, проте заснована на прототипах, а не на класах. Є чотири типи об’єктів: вбудовані об’єкти, об’єкти браузера, об’єкти документа і об’єкти користувача (програміста).

Введення-виведення в основному обмежене взаємодією з документами і користувачами. За умовчанням передбачається, що доступ до локальної файлової системи заборонений. Однак браузери можуть надавати спеціальні об’єкти, за допомогою яких забезпечується робота з файловою системою користувача, хоча і з видачею попереджень про небезпеку виконання файлових операцій.

Сценарії JavaScript активно взаємодіють з об’єктами, вбудованими в Web-сторінку. Для цього вони, власне, і створюються. Але перш, ніж ця взаємодія стане можливою, слід впровадити код сценарію в текст HTML-документа. Існує кілька способів зв'язати HTML-документ з конкретним сценарієм (скриптом), але зазвичай їх просто розміщують всередині контейнерного тега <SCRIPT>, тобто між дескрипторами <script> і </script>.

Контейнер <SCRIPT> в цьому випадку буде перебувати безпосередньо в HTML-документі, причому у довільному його місці. Програмний код пишуть прямо в HTML-документі або в спеціальних текстових файлах, які можна викликати з головного HTML-документа. Для початку розглянемо перший варіант. Перш за все браузер знаходить тег <script> в тілі веб-документа, і весь наступний текст намагається обробити як скриптовий код. І так до тих пір, поки не зустріне закриваючий тег </script>. Після цього всі наступні символи будуть вважатися HTML-текстом. Будь-який HTML-документ може містити довільне число «скриптових включень», але кожне має відкриватися і завершуватися відповідним тегом. Від їх розташування у тілі HTML-документа іноді може залежати функціонування всієї Web-сторінки, але про це буде сказано пізніше.

Контейнерний тег <script> може містити атрибут SRC, який вказує ім’я або URL-адресу текстового файлу, що містить код сценарію. Цей атрибут необхідний в тому випадку, якщо сценарій розташований не безпосередньо в HTML-документі, а в окремому файлі. Розширення файлу зі сценарієм може бути яким завгодно, але зазвичай використовують js:

```
<SCRIPT SRC = «myscripts.js»> </ SCRIPT>.
```

Якщо сценарій розташовується в окремому файлі, то в ньому, зрозуміло, теги <SCRIPT> і </ SCRIPT> не пишуть. Сценарій, завантажений з зовнішнього файлу, можна уявити собі просто як його вставку в HTML-документ.

У браузерах, що потенційно підтримують сценарії, цю функцію користувач може відключити (зокрема, із міркувань підвищеної безпеки). Крім того, існують браузери, які принципово не підтримують сценарії. У даній ситуації бажано хоча б вивести повідомлення про те, що на сторінці був сценарій, але в даному конкретному випадку він не виконується. З цією метою в HTML-документі використовують контейнерний тег <noscript>, в якому розміщують текст, який з’явиться користувачеві за умови, що сценарій з тієї, чи іншої причини не виконуватиметься. Всі браузери, які підтримують сценарії, проігнорують вміст тегів <noscript> крім тих випадків, коли підтримка сценаріїв відключена. Браузери, що принципово не підтримують сценарії, навпаки, вміст тега <script> опустять (особливо, якщо він вкладений у теги <!-- -->, які є HTML-коментарем), а тега <noscript> - відобразять. Приклад наведено нижче:

```
<HTML> <HEAD>
<TITLE> Приклад застосування тега NOSCRIPT </ TITLE>
</ HEAD>
<SCRIPT TYPE = "text / JavaScript»>
<! -
alert ( «Підтримка JavaScript включена»); // ->
</ SCRIPT>
<NOSCRIPT>
<H2> Ваш браузер не підтримує JavaScript або його підтримка відключена </ H2>
</ NOSCRIPT>
</ HTML>
```

Тут був використана функція alert (повідомлення) для виведення повідомлень в маленькому діалоговому вікні.

Як уже зазначалося, в окремих файлах зазвичай розміщують бібліотеки функцій (визначення функцій), а також сценарії, які використовуються в декількох HTML-документах одного або декількох сайтів. Сценарій можна також писати у вигляді рядка операторів, між якими ставиться крапка з комою. Такий рядок, взятий в лапки, служить у якості значення атрибута-події, наприклад:

```
<IMG SRC = "mypicture.gif" ONCLICK = "alert ('Привіт!')".
```

Виклики функцій і їх визначення можуть слідувати в довільному порядку, але тільки якщо вони розташовані в одному і тому ж контейнері <script>. Якщо вони розміщуються у різних контейнерах <script>, то необхідно, щоб визначення функції передувало її виклику. Аналогічно, якщо визначення функцій знаходяться в окремому файлі, то його необхідно завантажити в HTML-документ раніше викликів цих функцій.

При спробі завантажити і виконати неправильний варіант HTML-коду з'явиться діалогове вікно з повідомленням «Помилка: Передбачається наявність об'єкта». Браузер інтерпретує теги HTML послідовно. Так, зустрівши вираз виклику функції myfunc() в одному контейнері <script>, браузер ще не має в пам'яті визначення цього об'єкта (функції), розташованого в іншому контейнері <script>. Якщо ж визначення функції і її виклик знаходяться в одному і тому ж контейнері <script>, то його вміст спочатку завантажується в пам'ять, а потім аналізується. Коли інтерпретатор зустрічає виклик функції, то він шукає її визначення в пам'яті і в разі успіху виконує код цієї функції.

Якщо необхідно, щоб сценарій виявився в браузері перш, ніж буде завантажено елементи HTML-документа, то його слід розташувати у верхній частині HTML-коду, а ще краще в контейнері <head> в заголовку документа.

3.1.2.2. Засоби Back-end розробки.

Як зазначалося вище, під Back-end'ом мають на увазі програмні компоненти, що працюють на сервері, і вибір засобів для реалізації цих компонентів є вкрай широким – рис. 3.4, справа.

Умовно усі засоби для серверного веб-програмування можна поділити на два класи: окремі програми, що працюють відповідно до технології CGI та мови, висхідні коди яких вбудовуються прямо у веб-сторінку та проганяються препроцесором веб-серверу перед видачею у браузер клієнта. Другий підхід є більш зручним, про що свідчить все більше занепадання CGI-засобів. У якості альтернативи для них виступають такі серйозні продукти, як:

- серверні сторінки Java - JSP, що активно просуваються любителями цієї відкритої, сучасної мови програмування (загального призначення);

- активні серверні сторінки ASP від корпорації Microsoft (відповідно ступінь підтримки цього рішення надзвичайно високий);

- використання мови PHP, яку і обирає більшість розробників серверних додатків через цілий комплекс позитивних рис цього продукту, які розглянемо докладніше.

На сьогоднішній день PHP є найбільш поширеною мовою веб-програмування. Переважна більшість сайтів і веб-сервісів в Інтернеті написано за допомогою PHP. За деякими оцінками PHP застосовується більш ніж на 80 % сайтів, серед яких такі сервіси, як facebook.com, vk.com, baidu.com і інші. І така популярність не видається дивною. Простота мови дозволяє швидко і легко створювати сайти і портали різної складності.

PHP був створений в 1994 році датським програмістом Расмусом Лердорфом і спочатку являв собою набір скриптів на іншій мові програмування Perl. Пізніше цей набір скриптів був переписаний в інтерпретатор на мові C. Із самого виникнення PHP представляв зручний набір інструментів для спрощеного створення веб-сайтів і веб-додатків.

Розглянемо, які ж переваги надає PHP:

- для всіх найбільш поширених операційних систем (Windows, MacOS, Linux) є свої версії пакетів розробки на PHP, а це означає, що можна створювати веб-сайти на будь-якій з цих операційних систем;

- PHP може працювати в зв'язці з різними веб-серверами: Apache, Nginx, IIS, тощо;

- простота і легкість освоєння є особливістю цієї мови. Як правило, маючи навіть невеликий досвід в програмуванні на PHP, можна створювати простенькі веб-сайти;

- PHP схожий на мову C, тому, знаючи C, або одну з мов з C-подібним синтаксисом, буде простіше опановувати PHP;

- PHP підтримує роботу з великою кількістю систем баз даних (MySQL, MS SQL Server, Oracle, Postgres, MongoDB, та інші);

- поширеність хостингових послуг і їх дешевизна. Так як, як правило, хостингові компанії розміщують веб-сайти на PHP на веб-серверах Apache або Nginx, які працюють на одній з операційних систем сімейства Linux. І веб-сервери, і операційні системи на базі Linux безкоштовні, що знижує загальну вартість використання хостингу;

- постійний розвиток, адже PHP продовжує розвиватися, виходять все новіші версії, які несуть нові функції, адаптуючи мову програмування до нових викликів. І, як правило, перехід на нову версію не викликає ніяких труднощів;

- існує надзвичайно багато якісних Інтернет-ресурсів з підтримки процесу програмування на PHP, де можна задати власне питання і найскоріше отримати кваліфіковану відповідь.

Зважаючи на усі позитивні риси, приймаємо PHP у якості основного засобу Back-end розробки у даному проекті.

Таким чином, у якості мов програмування для обраного напрямку (веб-розробки) обираємо PHP та JavaScript.

Останнім питанням є вибір конкретного середовища розробки, що дозволяє писати код на цих обраних мовах. У якості такого середовища

можна взяти популярний на сьогоднішній день серед студентської спільноти Visual Studio Code, або більш професійний PHP Storm, або простий текстовий редактор типу Sublime Editor. Така широта вибору засобів обумовлена простотою задачі (зокрема, відсутністю необхідності застосування системи управління базами даних, і т.п.) та стандартністю (поширеністю) обраних мов програмування серед представників сучасної програмістської спільноти.

3.2. Особливості реалізації елементів обраних алгоритмів, пов’язаних із мінімізацією маршрутних даних.

Розглянемо особливості операцій 1-8 з рис. 2.7.

Операція 1, яка полягає у завантаженні файлу з нестиснутою маршрутною інформацією не має жодних особливостей, хіба що файл слід відкривати у текстовому режимі.

Операція 2 потребує зчитування усіх чисел із файлу і запису знову, але з меншою кількістю розрядів після десяткової коми (5 замість 6). Для форматованого виведення у файл можна, наприклад, використовувати функцію fprintf.

Операцію 3 при роботі з переміщеннями сільськогосподарської техніки по полю можна не виконувати, обираючи за умовчанням п. 4 із табл. 2.2.

Операція 4 виконуватиметься наступним чином:

- зчитується чергова координата X, і якщо її базові розряди не відповідають поточній базі по X, то у вихідний файл записується новий тег B_X;
- зчитується чергова координата Y, і якщо її базові розряди не відповідають поточній базі по Y, то у вихідний файл записується новий тег B_Y;
- розряди, що відносяться до бази, у всіх координат відкидаються у будь-якому випадку.

Виконання операції 5 слід вести за алгоритмом, блок-схема якого наведена на рис. 3.5, текстовий опис алгоритму наведемо нижче.

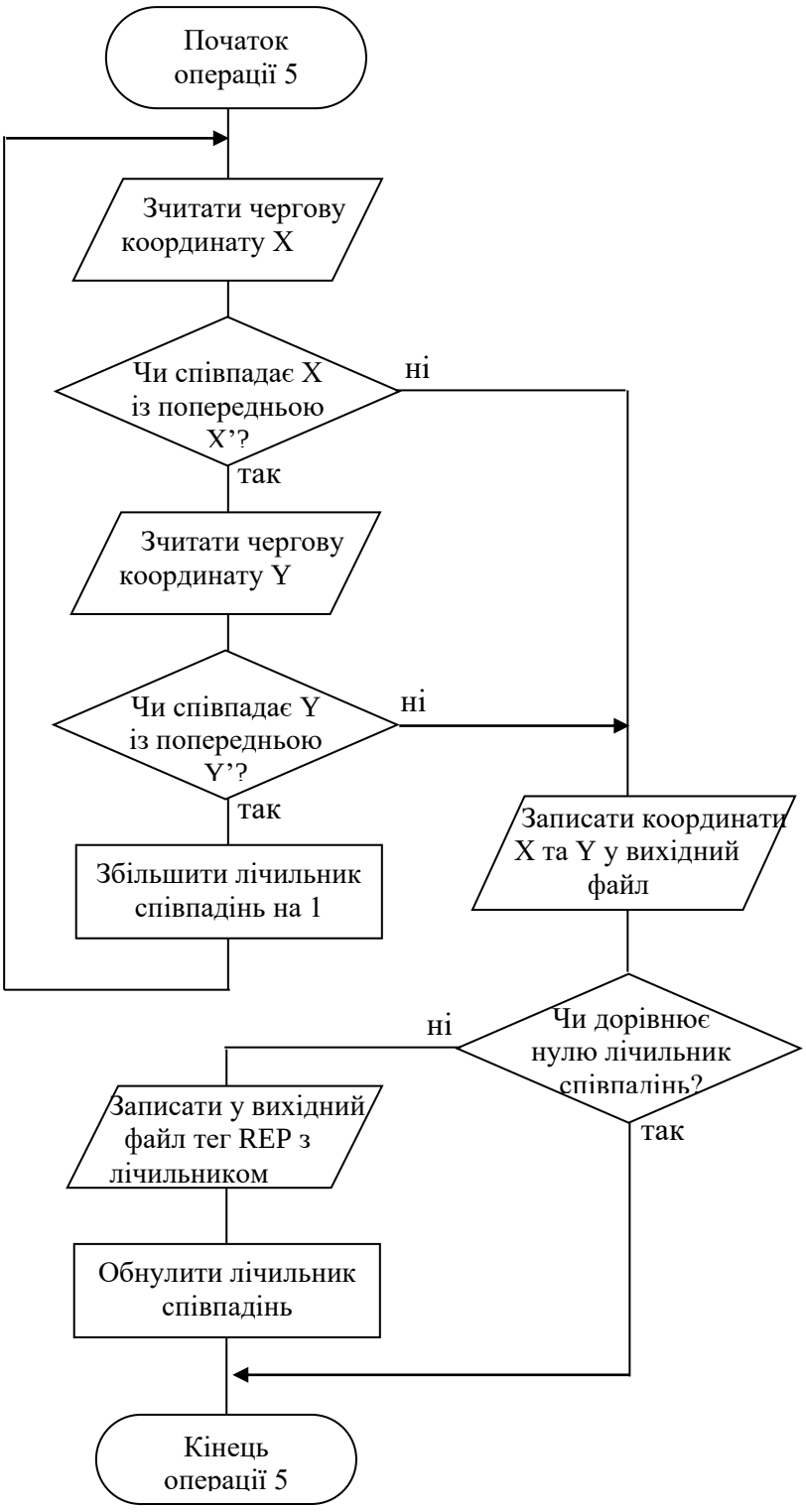


Рис. 3.5. Блок-схема виконання операції 5 загального алгоритму стиснення маршрутних даних, що полягає в урахуванні повторів (тег REP).

Так, по-перше, при зчитуванні чергової координати X слід порівнювати її із попереднім значенням X', і, якщо є співпадіння, порівнювати ще й координату Y із попередньою. Якщо співпали обидві координати, то ці

координати у вихідний файл не слід записувати, а треба збільшити на одиницю лічильник співпадінь; якщо ж співпадіння немає, то цей лічильник слід обнулити. Якщо при обнулінні виявилося, що значення лічильника від попередніх порівнянь не дорівнює нулю, то у вихідний файл слід внести тег REP із відповідною кількістю повторів.

При виконанні операції 6 достатньо всього лише зчитувати координати із вхідного файлу як рядки, а записувати у вихідний файл як числа відповідного формату (цілі, довжиною 1 байт). Ці дії, як і у пункті 2, можна виконувати за допомогою функції fprintf.

Операція 7 може бути виконана будь-яким стандартним способом архівування, наприклад, алгоритмом ZIP.

Операція 8 являє собою просте збереження отриманого файлу у потрібному місці на сервері.

Таким чином, бачимо, що реальну алгоритмічну складову мають тільки кроки 4 та (у більшій мірі) крок 5. В цілому, у підрозділі розглянуто особливості реалізації усіх 8 елементів алгоритму мінімізації маршрутних даних, розробленого у підрозділі 2.4.

3.3. Визначення формату зберігання маршрутної інформації.

Вище по тексту роботи уже чимало уваги було приділено особливостям зберігання маршрутної інформації, тому можна зробити висновок, що розроблено власний формат зберігання відповідної інформації. Розглянемо його докладніше.

По-перше, у кінцевому файлі на початку окремим рядком має бути записана дата та окремим рядком час початку запису маршрутної інформації. Після цього окремим рядком має бути вказаний інтервал, через який знімаються дані про місцеположення рухомого об’єкта, що відстежується – стандартно 1 секунда.

Далі в окремих рядках мають розміщуватися теги типу BASE, яких існує два різновиди:

- B_X – вказання бази для першої координати X;

- B_Y – вказання бази для другої координати Y.

Справа від цих тегів мають розміщуватися числові значення розрядів, що входять до відповідної бази, наприклад:

B_X50.382

B_Y38.405

Далі в окремих рядках мають розміщуватися пари координат X та Y, причому тільки їх молодші розряди (що не входять до баз), закодовані у числовій формі.

Якщо якісь координати повторюються, то у файлі буде введено тег REP (точніше його скорочена форма R), після якого указується кількість повторів. Тут *N* однакових рядків відповідають N–1 повторам, наприклад, трьом підряд однаковим рядкам координат відповідає тег виду:

R2

Таким чином, реалізуючи процес мінімізації маршрутної інформації вказаним шляхом, та зберігаючи дані в описаному форматі, можна досягти високих показників ефективності підсистеми стиснення, що в ідеалі має досліджуватися експериментальним шляхом, на основі працюючого програмного продукту, до опису реалізації якого слід перейти.

3.4. Особливості впровадження розробленої реалізації системи мінімізації маршрутних даних у конкретному технічному рішенні.

Для можливості поступового зведення, увесь код писався під керівництвом програмного продукту типу All-in-One, що називається Denwer і включає веб-сервер (Apache), СУБД (MySQL або MareaDB), інтерпретатор мови PHP, мови Perl, поштовий сервер, а також допускає використання портабельної версії, що завантажується прямо з флеш-пам’яті.

Реалізуємо усі розроблені вище алгоритми мовою PHP з використанням JavaScript та отримуємо результат, показаний у вікні рис. 3.6.

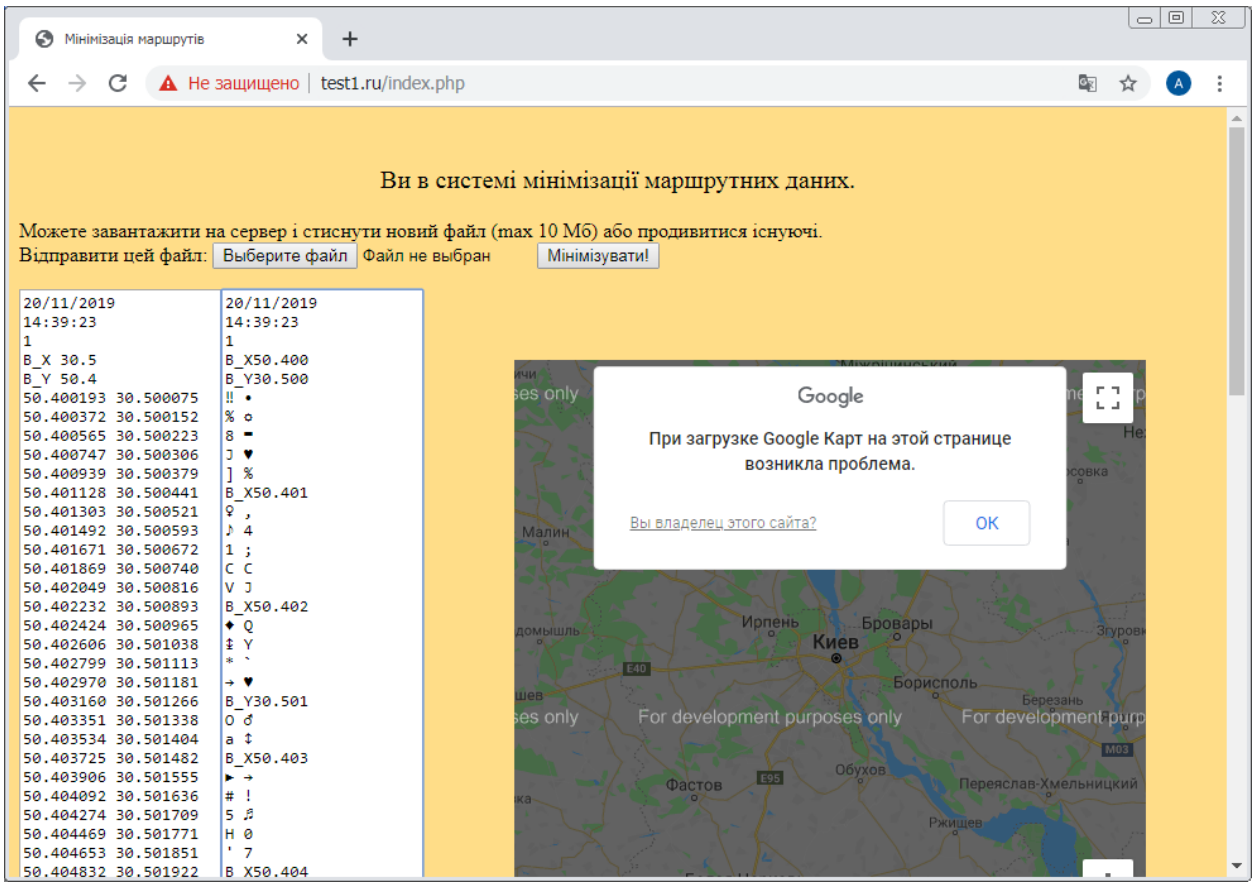


Рис. 3.6. Приклад стиснення файлу з маршрутною інформацією за допомогою розробленої системи.

Бачимо, що у систему можна завантажувати файли стандартною кнопкою «Выберите файл», після чого слід натиснути кнопку «Мінімізувати!». Обраний файл з локального комп'ютера відправляється на сервер і там стискається, результат чого відображується у правому вертикальному текстовому полі типу textarea, а головне, передається в архівну папку для постійного зберігання.

У правій частині вікна програми бачимо спробу відображення маршруту на Гул картах, однак із середини 2018 року сервіс надається на платній основі. Хоча система і надає 200 у.о. безкоштовного доступу на місяць, однак все одно для реєстрації потребує введення реальних даних кредитної карти і саме найгірше те, що якщо якимось чином по зареєстрованому аканту буде здійснено запитів більше, ніж на 200 доларів у місяць (а це 20 тис. запитів на місяць), то різниця буде автоматично списана з

рахунку підписанта. Таким чином, за умови створення атаки типу DDoS, сайт, що використовує Google Maps API автоматично в кінці місяця може стати винним компанії Google тисячі доларів (чи навіть більше). Таким чином, використовувати цю систему для некомерційних ресурсів на даному етапі її розвитку просто небезпечно з фінансової точки зору.

В цілому ж можна сказати, що система мінімізації маршрутної інформації реалізована і готова до практичного використання.

3.5. Аналіз результатів роботи системи.

Очевидним є факт стиснення маршрутної інформації за допомогою розробленої системи, однак ефективність цього процесу може бути різною, в першу чергу в залежності від обраного варіанту представлення бази: п.2 з табл. 2.2 або п.4, який у попередніх розділах і прийнято за основний варіант. На рис. 3.6 наведено приклад використання бази із 3 цифрами після десяткової коми і в результаті цього файл із маршрутною інформацією, що мав початковий розмір 2149 Кб було стиснено до об’єму 901 Кб, що складає біля 42% від початкового об’єму, а після застосування архівації об’єм зменшився ще майже вдвічі і склав 487 Кб. Загальний ступінь стиснення складає біля 22% від початкового об’єму, що майже на 10% краще, ніж застосування простого архівування.

Таким чином, система, розроблена в рамках даної роботи (тобто без залучення великих фінансових інвестицій) показує достатній рівень економічної ефективності і дозволяє економити дисковий простір не менше ніж на 10%, що при великих обсягах інформації, що зберігається (як у централізованому файловому архіві, що і описаний у даній роботі) приводить до аналогічної економії фінансів на постійні запам’ятовуючі пристрої.

3.6. Висновки по розділу.

Таким чином, у розділі описано проектування та реалізацію системи мінімізації маршрутної інформації, яку виконано у вигляді веб-додатку.

Такий варіант є зручним з організаційної точки зору, оскільки користувач із будь-якої точки планети через Інтернет може підключитися до розробленої системи і завантажити туди файл з маршрутною інформацією, який автоматично буде стиснуто, результат показано користувачеві, а сам стиснутий файл вміщується у папку з архівами. Створена система дає економію дискового простору, як мінімум на 10% більше, ніж застосування простого архівування, тому її доцільно застосовувати у практичній діяльності.

ВИСНОВКИ

У даній роботі розроблено систему мінімізації маршрутної інформації на основі оригінального розробленого алгоритму. В роботі проаналізовано існуючі способи зняття, перетворення та зберігання маршрутної інформації з оглядом на використання можливостей зменшення її обсягів. Після цього на основі відомих алгоритмів розроблено комплексний оригінальний алгоритм мінімізації маршрутних даних, а також виконано необхідні супутні науково-технічні рішення (як, наприклад, моделювання маршрутної інформації, створення власного формату збереження даних та ін.).

Також здійснено програмну реалізацію розробленого алгоритму та її дослідження ефективності, що показало покращення показників стисливості мінімум на 10% у порівнянні з традиційними методами. В роботі враховано різноманітні залежності (зокрема, від швидкості руху об’єкта), розглядаються вимоги надійності збереження, захисту відповідної маршрутної інформації.

Розроблений програмний продукт функціонує і може бути використаний в реальних прикладних задачах.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Flanagan D. JavaScript: The Definitive Guide. 7th ed. Sebastopol: O’Reilly Media, 2020. 706 p.
2. Duckett J. JavaScript and JQuery: Interactive Front-End Web Development. Indianapolis: Wiley, 2014. 640 p.
3. Zakas N. Professional JavaScript for Web Developers. 4th ed. Indianapolis: Wrox, 2021. 960 p.
4. Freeman E., Robson E., Bates B., Sierra K. Head First HTML and CSS. 2nd ed. Sebastopol: O’Reilly Media, 2012. 768 p.
5. Meloni J. Sams Teach Yourself PHP, MySQL & JavaScript. 7th ed. Indianapolis: Pearson Education, 2017. 720 p.
6. Nixon R. Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5. 6th ed. Sebastopol: O’Reilly Media, 2018. 829 p.
7. Powers D. PHP Solutions: Dynamic Web Design Made Easy. 5th ed. New York: Apress, 2023. 592 p.
8. Teixeira M. Pro PHP Programming. New York: Apress, 2014. 508 p.
9. Glassford J. PHP 8 Objects, Patterns, and Practice. 6th ed. New York: Apress, 2021. 624 p.
10. Resig J., Bibeault B., Kats E. Secrets of the JavaScript Ninja. 2nd ed. Shelter Island: Manning Publications, 2016. 464 p.
11. Crockford D. JavaScript: The Good Parts. Sebastopol: O’Reilly Media, 2008. 176 p.
12. McFarland D. CSS: The Missing Manual. 5th ed. Sebastopol: O’Reilly Media, 2017. 688 p.
13. Keith J., Andrew R. HTML5 for Web Designers. 2nd ed. New York: A Book Apart, 2016. 176 p.
14. Osmani A. Learning JavaScript Design Patterns. 2nd ed. Sebastopol: O’Reilly Media, 2017. 254 p.

15. Chaffer J., Swedberg K. Learning jQuery. 4th ed. Birmingham: Packt Publishing, 2013. 444 p.

16. Fowler S. Web Development with Node and Express: Leveraging the JavaScript Stack. Sebastopol: O'Reilly Media, 2019. 480 p.

17. Beighley L. Head First PHP & MySQL. Sebastopol: O'Reilly Media, 2008. 812 p.

18. Bovet D., Ciscon S. Understanding Linux Web Hosting. Boston: Addison-Wesley, 2016. 320 p.

19. Іванчук С., Петренко В. Основи Web-програмування: навчальний посібник. Львів: Видавництво Львівської політехніки, 2020. 210 с.

20. Ковальчук А. Сучасні технології Web-розробки: навч. посіб. Київ: Кондор, 2019. 198 с.