МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проекту (роботи)
магістра
(ступінь вищої освіти (освітній ступінь))

на тему **КОМП'ЮТЕРНА СИСТЕМА ДЛЯ МІНІМІЗАЦІЇ МАРШРУТНИХ ДАНИХ ДЛЯ СИСТЕМ НАКОПИЧЕННЯ**

Виконав: студент 2 курсу, групи КНТ-522м
спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні системи та мережі

БЕЛЬСЬКИЙ Т.А.
(ПРІЗВИЩЕ та ініціали)
Керівник ІЛЬЯШЕНКО М. Б.
(ПРІЗВИЩЕ та ініціали)
Рецензент ЄФИМЕНКО М.В.
(ПРІЗВИЩЕ та ініціали)

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти магістерський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) «Комп'ютерні системи та мережі»
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Зав. кафедри Кудерметов Р.К.
_____
"____" _____ 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА

БЕЛЬСЬКОГО Тимофія Володимировича
(прізвище, ім'я, по батькові)

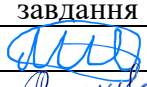
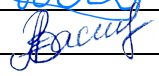
1. Тема проекту (роботи) Комп'ютерна система для мінімізації маршрутних даних для систем накопичення
керівник проекту (роботи) ІЛЬЯШЕНКО Матвій Борисович, к. т. н., доцент
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від " 24 " жовтня 2023 року № 400
2. Строк подання студентом проекту (роботи) 10 грудня 2023 року

3. Вихідні дані до проекту (роботи) наявні методи та алгоритми мінімізації маршрутних даних, точність визначення широти і довготи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1) Аналіз технічного завдання;
2) Проектування систем;
3) Дослідження системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проекту (роботи)

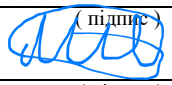
Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	ІЛЬЯШЕНКО М. Б., доцент		
нормоконтроль	ПОЛЬСЬКА О.В., ст. викл.		

7. Дата видачі завдання 01.10.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз існуючих підходів до вирішення	14.10.2023 р.	
	проблем зберігання маршрутної інформації		
2	Розробка алгоритмів мінімізації маршрутних даних	17.10.2023 р.	
3	Вибір засобів розробки	23.10.2023 р.	
4	Розробка системи мінімізації маршрутних даних	16.11.2023 р.	
5	Дослідження розробленої системи	27.11.2023 р.	
6	Оформлення отриманих результатів у ПЗ	04.12.2023 р.	
7	Оформлення графічного матеріалу	09.12.2023 р.	
8	Оформлення допоміжного матеріалу	10.12.2023 р.	

Студент  Тимофій БЕЛЬСЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)  Матвій ІЛЬЯШЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ

РЕФЕРАТ

ПЗ: 83 с., 18 рис., 2 табл., 16 джерел.

MATHCAD, PHP, JAVASCRIPT, МІНІМІЗАЦІЇ ДАНИХ.

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

Предмет дослідження – є методи та моделі мінімізації маршрутних даних.

Метою магістерської роботи – є ефективне скорочення обсягу маршрутних даних, що підлягають зберіганню у відповідних архівах, для забезпечення комплексної безпеки процесу переміщення рухомих об'єктів.

У роботі використано загальнонаукові методи аналізу (для вивчення наявних рішень і принципів їхньої роботи) і синтезу (для створення оригінальних рішень). Для опису інформаційної кривої траєкторії використовували методи вищої математики, а для реалізації програмного забезпечення та розробки всіх необхідних елементів - методи галузі інформаційних технологій.

В роботі отримано нові науково-технічні результати:

– для отримання маршрутної інформації було розроблено процедури моделювання;

– проаналізовано можливість використання методів вищої математики (сплайн-інтерполяція, інтегральні перетворення) для стиснення маршрутної інформації сільськогосподарської техніки (з урахуванням специфіки такої інформації);

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

– створено власний формат зберігання маршрутної інформації.

Робота реалізована у вигляді практичного програмного продукту, може зіграти важливу роль на практиці, наприклад, використовуватися в системах мінімізації маршрутної інформації в централізованих файлових архівах.

ABSTRACT

PP: 83 p., 18 figures, 2 tables, 16 sources.

MATHCAD, PHP, JAVASCRIPT, DATA MINIMIZATION.

The object of research is the process of accumulation and minimization of route data of moving objects.

The subject of research is the methods and models of route data minimization.

The purpose of the master's thesis is to effectively reduce the amount of route data to be stored in the relevant archives to ensure comprehensive safety of the process of moving objects. The paper uses general scientific methods of analysis (to study existing solutions and the principles of their operation) and synthesis (to create original solutions).

Methods of higher mathematics were used to describe the information curve of the trajectory, and methods of the information technology industry were used to implement the software and develop all the necessary elements.

New scientific and technical results were obtained in the work:

– modeling procedures were developed to obtain route information;

– analyzed the possibility of using methods of higher mathematics (spline interpolation, integral transformations) to compress the route information of agricultural machinery (taking into account the specifics of such information);

– based on scientific analysis, a new comprehensive algorithm for compressing route information was proposed;

– a proprietary format for storing route information was created.

The work is implemented in the form of a practical software product and can play an important role in practice, for example, it can be used in systems for minimizing route information in centralized file archives.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

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, PHP Hypertext Preprocessor;

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 пов'язаних з нею інформаційно-комунікаційних технологій призвів до численних змін у різноманітних галузях життя, як окремих людей, так і суспільства в цілому. Зокрема, суттєво змінилися засоби та відповідні підходи до забезпечення безпеки різних систем і сфер нашого життя. Одним із прикладів є широке поширення систем відеоспостереження ("CCTV"). При цьому зі збільшенням кількості камер їхня робота стає дедалі більш автоматизованою. Камери записують усю інформацію в архів, і, як правило, ніхто не бачить поточних зображень, і тільки за необхідності з цих безмовних цифрових архівів витягуються потрібні фрагменти відео. Проблема в тому, що в одному місті встановлюються десятки, сотні або тисячі камер, в одній країні - десятки тисяч, і зберігається величезна кількість цифрових даних. Точно з такою самою проблемою стикаються й інші системи безпеки, наприклад, системи, що відстежують поточне місце розташування рухомих транспортних засобів.

Такі системи, наприклад СВС, мають спеціальні налаштування, що дають змогу зменшити обсяг даних, що записуються, але це може бути зроблено тільки за рахунок погіршення "якості" інформації, що зберігається, за конкретною предметною галуззю. Деяка "деградація" інформації до певної межі допустима, але, навпаки, вміст деяких даних у такій інформації може змінитися більш ніж на кілька десятків відсотків (що абсолютно неприпустимо). Тому важливим і актуальним завданням є розроблення методів, що дають змогу одночасно "безпечно" й ефективно мінімізувати дані про траєкторії руху керованих об'єктів.

Для досягнення поставленої мети необхідно вирішити наступні задачі дослідження:

– аналіз наявних методів збору, перетворення та зберігання маршрутної інформації та використання можливостей щодо скорочення їх обсягу;

– розробити алгоритми мінімізації маршрутної інформації на основі відомих алгоритмів та оптимізувати їх;

– здійснити програмну реалізацію розроблених алгоритмів та їх синтез;

– використовувати особливості процесу мінімізації з урахуванням різних залежностей (зокрема, швидкості руху об'єкта), вимог до надійності зберігання та захисту відповідної маршрутної інформації.

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Сфери застосування маршрутної інформації

Зручність процесу зберігання маршрутної інформації та її широке й різноманітне застосування в сучасних системах захисту, контролю та управління в загальних рисах було розглянуто вище. Розглянемо докладніше, де саме і для чого використовується ця маршрутна інформація.

Перш за все, системи моніторингу поточного місця розташування (СКПМ) використовуються для спеціалізованої рухомої техніки. У цьому випадку цілком природно, що власник (або його агент, або інша уповноважена особа) хоче мати можливість перевірити, чи використовується його транспортний засіб строго за призначенням. Перш за все, це стосується положення рухомого складу в машині, наприклад, у таких трьох випадках.

1) Поточне місце розташування муніципальних автобусів може перевірити будь-який житель населеного пункту, якому належить техніка. Наразі це реалізовано у вигляді спеціальної веб-сторінки, на якій будь-який користувач Інтернету може вільно переглянути місце розташування транспорту, яке відстежують GPS-трекери і датчики ГЛОНАСС;

2) Місцезнаходження спеціальної снігоприбиральної техніки в зимовий період. Спеціальна снігоприбиральна техніка, як і звичайні автомобілі, оснащена GPS-модулем, і будь-хто охочий може перевірити місцезнаходження всіх машин на сайті Дорожньої служби (або аналогічному), оцінити ефективність їхнього використання, поскаржитися або подякувати керівництву організації;

3) Завдання відстеження місцезнаходження сільськогосподарської техніки її власником (або уповноваженою особою, що діє від імені власника) є вкрай важливим для України, великої сільськогосподарської країни. Адже, на жаль, на практиці непоодинокими є випадки, коли власники техніки нехтують проведенням робіт, зливають пальне, а потім продають машину, стверджуючи, що роботи виконано, а пальне використано за призначенням. При цьому поля залишаються

неполитими або неораними. Така ситуація, звісно ж, неприпустима, оскільки вкрай негативно позначається на продуктивності всього сільськогосподарського виробництва.

Для більшої точності ситуація, коли рухома техніка переміщується за заданим маршрутом не з волі оператора (водія), а використовується (або не використовується) не за призначенням, називається "несанкціонованим використанням рухомої техніки" (НВРТ).

Розглянемо останній випадок детальніше, оскільки він має важливі особливості порівняно з іншими описаними вище ситуаціями. Інформація про маршрут потрібна не одразу, а лише через кілька годин, наступного дня чи навіть тижня. У той же час, наприклад, коли пасажири чекають на автобусній зупинці, їм потрібна актуальна інформація про поточне місцезнаходження громадського транспорту в даний момент, і ця інформація стає застарілою відразу після того, як вони здійснили поїздку або сіли в автобус. Так само інформація про місцезнаходження снігоприбиральної техніки має першочергове значення саме в цей момент. Адже при плануванні маршруту важливо знати, яка дорога приведе їх додому, або де зараз працює рятувальне обладнання, якщо людина потрапила під лавину, але ця інформація втрачає свою цінність для них, як тільки вони покидають засніжену територію.

Так, маршрутна інформація про переміщення сільськогосподарської техніки відрізняється тим, що вона повинна зберігатися протягом певного часового інтервалу (не менше тижня) і тому потребує відповідного пристрою зберігання (архіву), який, звісно, коштує певних грошей. Очевидно, що збільшення часового інтервалу зберігання маршрутних даних збільшує вартість системи і знижує загальну економічну ефективність. З іншого боку, бездумне зменшення цього параметру збільшує ймовірність невиявлення НВРТ, що може призвести до вищезгаданих збитків на підприємстві.

Збільшення часу зберігання призводить до зростання вартості цифрових носіїв для архівування, а зменшення цього часу ΔT - до збільшення втрат часу зберігання. Очевидно, що вибір оптимального значення ΔT є свого роду

математичною оптимізаційною задачею, проте в ній присутній і стохастичний елемент, оскільки передбачити НВРТ будь-якого водія доволі складно (ймовірність цього може змінюватися залежно від поточних умов життя людини, причому навіть для одного й того самого водія Оскільки це навіть можливо, то, звісно, краще мінімізувати таку можливість та інформувати/нагадувати водія про необхідність постійного контролю за поточним положенням автомобіля на регулярній основі). Обґрунтування вибору конкретних значень для ΔT наведено у відповідному розділі цієї роботи.

Після того як для виконання сільськогосподарського завдання (наприклад, поливу на пересіченій місцевості) було прокладено зручний і безпечний маршрут, його може бути повторено іншим разом іншими водіями на основі збереженої маршрутної інформації "першопрохідця", який витратив час на пошук оптимального маршруту на пересіченій місцевості. Однак загалом такий варіант використання маршрутної інформації є набагато екзотичнішим (хоча й можливим на практиці), і його основне застосування все ж таки полягає в контролі за правильністю та дозволом використання рухомої сільськогосподарської техніки відповідно до загальних потреб її власника, а не ситуативних бажань оператора (водія).

1.2 Оцінка способів отримання маршрутної інформації

Таким чином, у попередньому розділі було встановлено, що маршрутна інформація широко використовується для цілей управління, тобто для забезпечення загальної безпеки рухомої техніки і всієї системи, до якої він включений. Наступним кроком є оцінка кількості такої інформації (вибір конкретних оптимальних значень ΔT), але для цього спершу необхідно зрозуміти, як отримується маршрутна інформація, тобто зрозуміти її природу.

Насамперед, розглядаючи об'єкти на поверхні Землі, важливо пам'ятати, що наша планета практично сферична (хоча й дещо стисла, але для наших цілей це не так важливо). Тому будь-яка точка на її поверхні може бути охарактеризована двома числами: широтою та довготою - ці величини є кутами, що:

– утворює радіус-вектор (географічну широту) між точкою на поверхні Землі та екваторіальною площиною;

– одну з них утворено діаметральною площиною, що проходить через задану точку і вісь обертання Землі, іншу - площиною спеціального меридіана, що проходить через вісь обертання Землі і точку, довготу якої прийнято за нуль (грінвічський меридіан), тобто географічну довготу.

Точність визначення широти і довготи в сучасних навігаційних системах настільки висока, що точки, які лежать на земній поверхні, можна розрізнити на відстані приблизно одного метра.

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

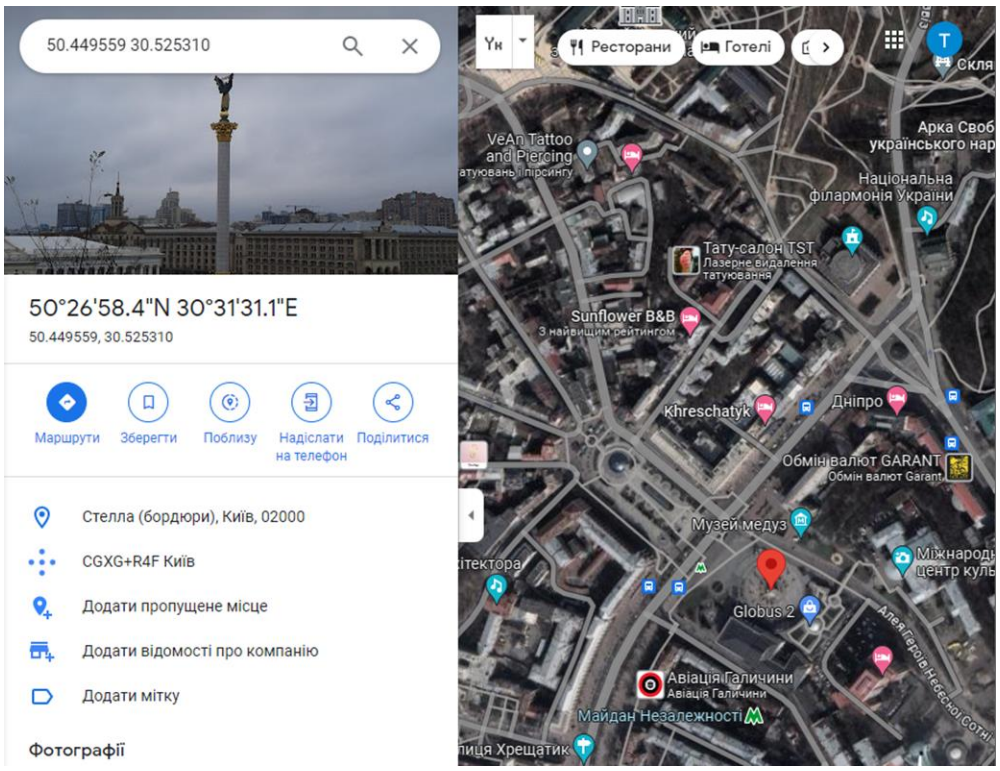


Рисунок 1.1 - Приклад завдання координат точки на карті Google.

Як видно, запис координат схожий на запис дробів із введенням додаткових символів, таких як апострофи та знаки позначки градуса:

$$50°26'58.4"N\ 30°31'31.1"E \quad (1.1)$$

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 точка може бути представлена десятковою часткою:

$$50.449559\ 30.525310 \quad (1.2)$$

Тут перше число, широта, може змінюватися від -90 до 90 градусів (позитивні значення відповідають "півночі"), а друге число, довгота, може змінюватися від -180 до 180 градусів (позитивні значення відповідають східній довготі, а негативні - західній). Перехід від однієї форми позначення координат до іншої простий, якщо згадати, що в одному градусі 60 хвилин, а в одній квадратній хвилині - 60 квадратних секунд. Наприклад, перехід від (1.1) до (1.2) здійснюється таким чином:

$$50 + 26/60 + 58{,}4/3600 \approx 50.449556,$$
$$30 + 31/60 + 31{,}1/3600 \approx 30.525306.$$

Зворотне перетворення виконується у два етапи (спочатку обчислюються хвилини, потім секунди):

$[0{,}449559/(1/60)] = 26'$

$[(0{,}449559 - 26/60)/(1/3600)] = 58{,}4"$

$[0, 525306/(1/60)] = 31'$

$[(0, 525306 - 31/60)/(1/3600)] = 31{,}1"$

Інакше кажучи, формат координат (1.2) є первинним у системі Google і перетворюється в (1.1) тільки в деяких випадках, щоб бути "читабельним" для людини. Формат (1.2) також є вдалим вибором з погляду об'єму пам'яті, необхідної для зберігання даних, оскільки для запису координат у цьому форматі потрібно 20 символів (включно з комою і пробілом або переведенням рядка), тоді як для запису у форматі "градуси-хвилини-секунди" потрібна більша кількість символів (26, включно з усіма розділювачами й уточнювальними символами). Формат (1.2) є кращим, тому надалі розглядається тільки текстове введення координат у форматі.

Найпростішим способом зберігання інформації є запис безпосередньо в текстовий файл типу протокол (log). Журнал матиме такий вигляд, як показано на рис. 1.2.

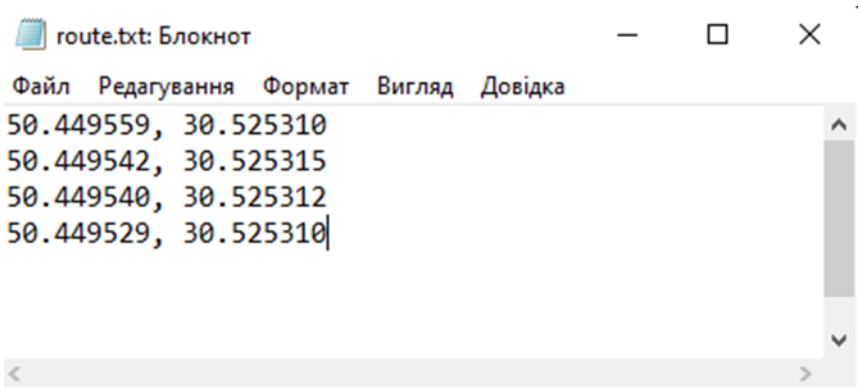


Рисунок 1.2 - Приклад найпростішої конфігурації маршруту

Найпростіший спосіб зберігання інформації, на жаль, є найбільш надлишковим. Оскільки інформація зберігається на цифрових носіях, нескладно реалізувати якусь систему кодування, що дає змогу істотно скоротити обсяг збереженої інформації. Такий підхід аналогічний стисненню без втрат і більш детально розглядається в наступному розділі. Аналогічно до схем стиснення з втратами під час обробки мультимедійних даних, під час зберігання інформації про маршрутизацію також можна відкидати частину інформації без шкоди для якості процесу контролю і запобігання НВРТ. Цей варіант, що дає змогу зменшити обсяг архівних даних, також описано в наступних підрозділах. Наразі актуальним є

визначення обсягу такої маршрутної інформації в найбільш невигідному, з погляду часу, поданні.

Отже, з аналізу рисунку 1.1, рівняння (1.2) та рисунку 1.2 видно, що для зберігання пари координат однієї точки на карті потрібно 20 символів. Іншими словами, якщо використовується кодування Unicode (2 байти на символ), загальна кількість байт буде наступною:

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

Тепер потрібно вирішити, як часто треба зберігати поточні координати. Існує два способи створення інформації про маршрут з інтервалами видачі:

– на регулярній основі;

– через рівні ділянки шляху.

Перший варіант технічно набагато простіший у реалізації. Це пов'язано з тим, що поточні координати визначають і передають в архів через заданий час без перевірки будь-яких додаткових умов (зокрема, без знання або врахування швидкості руху об'єкта).

Важливу роль у цьому підході відіграє вибір часового інтервалу Δt між фіксаціями координат. Його відповідне значення може бути оцінено з таких міркувань:

– швидкість руху комбайна під час збирання коливається від 5 до 15 км/год (наприклад, під час збирання соняшнику);

– максимальна швидкість комбайна під час перегонів становить близько 30 км/год;

– відстань між послідовними фіксаціями координат для протидії НВРТ становить близько 10 м;

– для запису правильного маршруту на частково хвилястій (або нерівній) місцевості відстань між послідовними корекціями координат має становити близько 2 м.

Тому для визначення відповідного значення Δt можна зробити такі дві оцінки:

– на перегонах:

$\Delta t1$ = 10 м / 30 км/год = 10 м / 8,3 м/с ≈ 1,2 с.

– під час руху по складній геометрії:

$\Delta t1$ = 2 м / 5 км/год = 2 м / 1,4 м/с ≈ 1,4 с.

Як видно, у кардинально різних ситуаціях необхідний інтервал часу між отриманням маршрутної інформації насправді доволі близький: ввівши невеликий запас у 20-40%, можна прийняти (і психологічно, і обчислювально) зручну цифру на рівні Δt:

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

Тому для подальшого аналізу фіксації положення на фіксованому інтервалі величина цього інтервалу береться відповідно до (1.4).

У процесі аналізу стають очевидними принципові недоліки такого підходу. Особливо яскраво це проявляється під час реєстрації положення об'єкта, що перебуває в нерухомому стані протягом тривалого часу. Якщо, як описано вище, положення реєструється кожну Δt=1 секунду, то за одну годину "стояння" (1 година = 3600 секунд) буде $n1=3600/\Delta t=3600$ однакових (тобто фактично надлишкових) записів. З урахуванням (1.3) кількість надлишкової інформації на годину буде такою:

$$B_1 = n1 \cdot \Delta b = 3600 \cdot \Delta b = 3600 \cdot 40 = 144000 \text{ б} = 140 \text{ Кб.} \qquad (1.5)$$

У разі використання іншого підходу, тобто даних про швидкість об'єкта, зокрема про те, що координати в архіві не можуть перетинатися, якщо швидкість дорівнює нулю (правило), кількість байт, записаних за весь час простою, дорівнює просто Δb (1.3).

У робочий час (за умови 12-годинної зміни) перший підхід дає такі результати:

$$В_{зм} = 12В_1 = 12 \cdot 140 \text{ Кб} = 1680 \text{ Кб} = 1{,}65 \text{ Мб}. \quad (1.6)$$

З одного боку, ця цифра не дуже велика, але впродовж сезону загальний обсяг інформації по одному комбайну може сягати кількох сотень мегабайтів, що за використання відповідних методів може бути зменшено в кілька разів. Таким чином, можна зменшити ємність, а отже, і кількість пристроїв зберігання даних та підвищити економічну ефективність усієї системи. Таким чином, бажано проаналізувати весь процес генерації та зберігання маршрутної інформації та застосувати спеціальні методи для зменшення її обсягу, тому перейдемо до розгляду цього питання.

1.3 Наявні методи мінімізації маршрутних даних

У попередньому підрозділі було описано процес генерації та архівування маршрутних даних, оцінено обсяг інформації, що генерується, та показано можливі шляхи вдосконалення процесу в цілому. До них відносяться методи, які генерують маршрутні дані з параметрами, що залежать від обсягу архівованих даних без втрат, з втратами та з параметрами, що залежать від швидкості транспортного засобу, що рухається.

Насамперед (метод, який використовується на практиці), кореневі дані, в якому б форматі вони не зберігалися, можуть бути стиснуті без втрат за допомогою звичайного програмного забезпечення для архівування. Це дає змогу заощадити значний дисковий простір.

По-перше, необхідно отримати велику кількість інформації про маршрут, але за відсутності фактичних файлів DATA це можна зробити за допомогою інтелектуального моделювання. Довжина дуги визначається як простий добуток радіуса на центральний кут і має бути виражена в радіанах:

$$s = \alpha \cdot R \qquad (1.7)$$

Якщо радіус Землі оцінюється в 6370 км, то, використовуючи (1.7), можна оцінити відстань, еквівалентну одній мільйонній градуса на поверхні Землі:

$$(0{,}000001^{\circ} \cdot \pi/180) \cdot 6370000 \approx 0{,}1 \text{ м}, \qquad (1.8)$$

Таким чином, одна частина на 100 000 відповідає відстані приблизно в один метр:

$$(0{,}00001^{\circ} \cdot \pi/180) \cdot 6370000 \approx 1 \text{ м}, \qquad (1.9)$$

Відстань в 1 км відповідає куту приблизно в 1/100 градуса:

$$(0{,}01^{\circ} \cdot \pi/180) \cdot 6370000 \approx 1000 \text{ м}, \qquad (1.10)$$

Відстань у 10 км відповідає куту приблизно в одну десяту градуса:

$$(0{,}1^{\circ} \cdot \pi/180) \cdot 6370000 \approx 10 \text{ км}, \qquad (1.11)$$

Іншими словами, якщо середній розмір поля становить близько 1 км, то різниця між кутовими координатами точок, розташованих навпроти його крайніх точок, дорівнює 1/100 градуса. На основі цієї інформації можна змоделювати великий обсяг маршрутної інформації, наприклад, у середовищі MathCad, використовуючи такі залежності:

$$i := 0 .. 500000$$

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

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

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

Тут як крайні точки поля використано координати 30,5º східної довготи і 50,4º північної широти (зліва внизу).

Отриманий набір координат близький до реальних координат, як показано на рис. 1.4, а. Така процедура моделювання необхідна для того, щоб правильно оцінити ступінь стиснення файлу координат. Наприклад, на рис. 1.4 б, показано файл практично такого самого розміру, але отриманий простим копіюванням тих самих координат. Як показано нижче, процедуру моделювання необхідно було виконати для того, щоб одержати реальне уявлення про можливі коефіцієнти стиснення, тому що коефіцієнти стиснення були б зовсім різними.

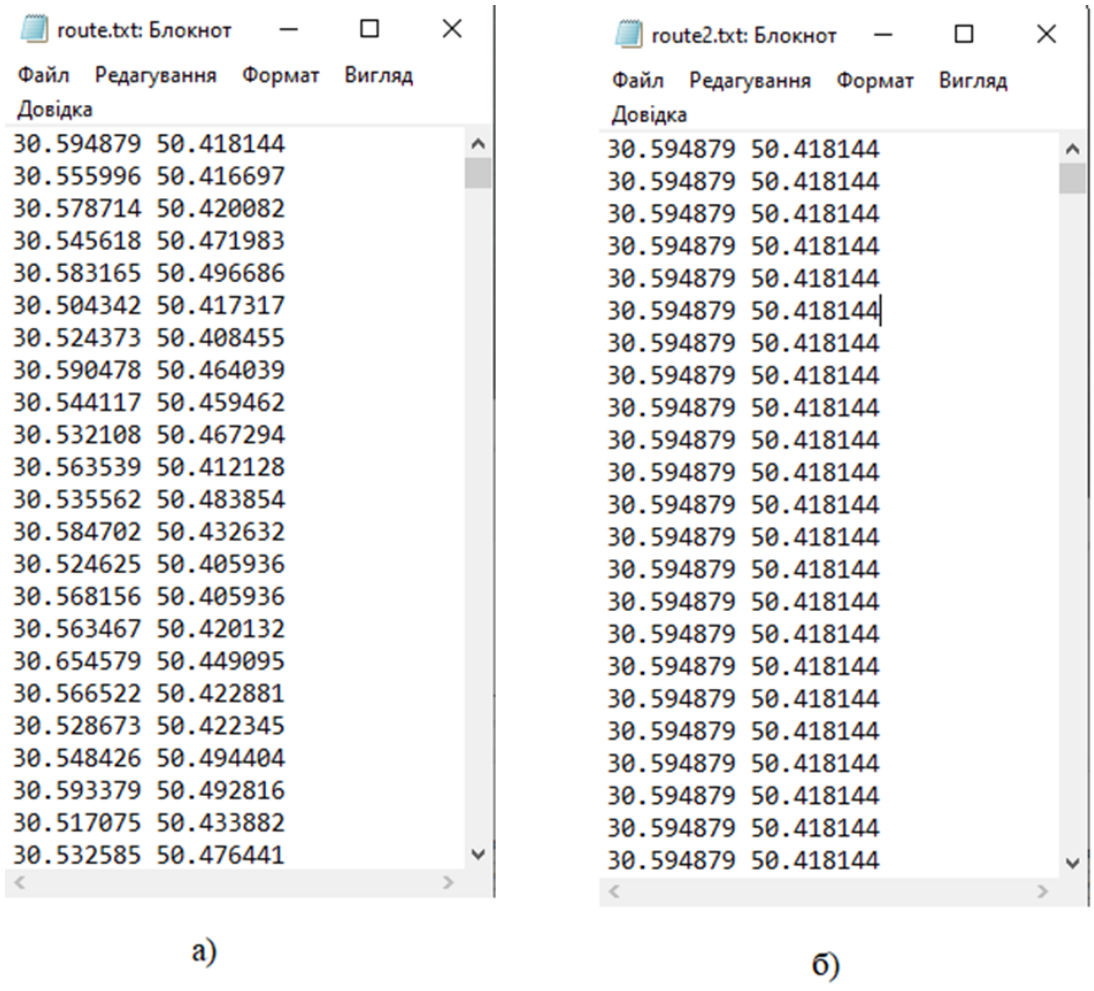
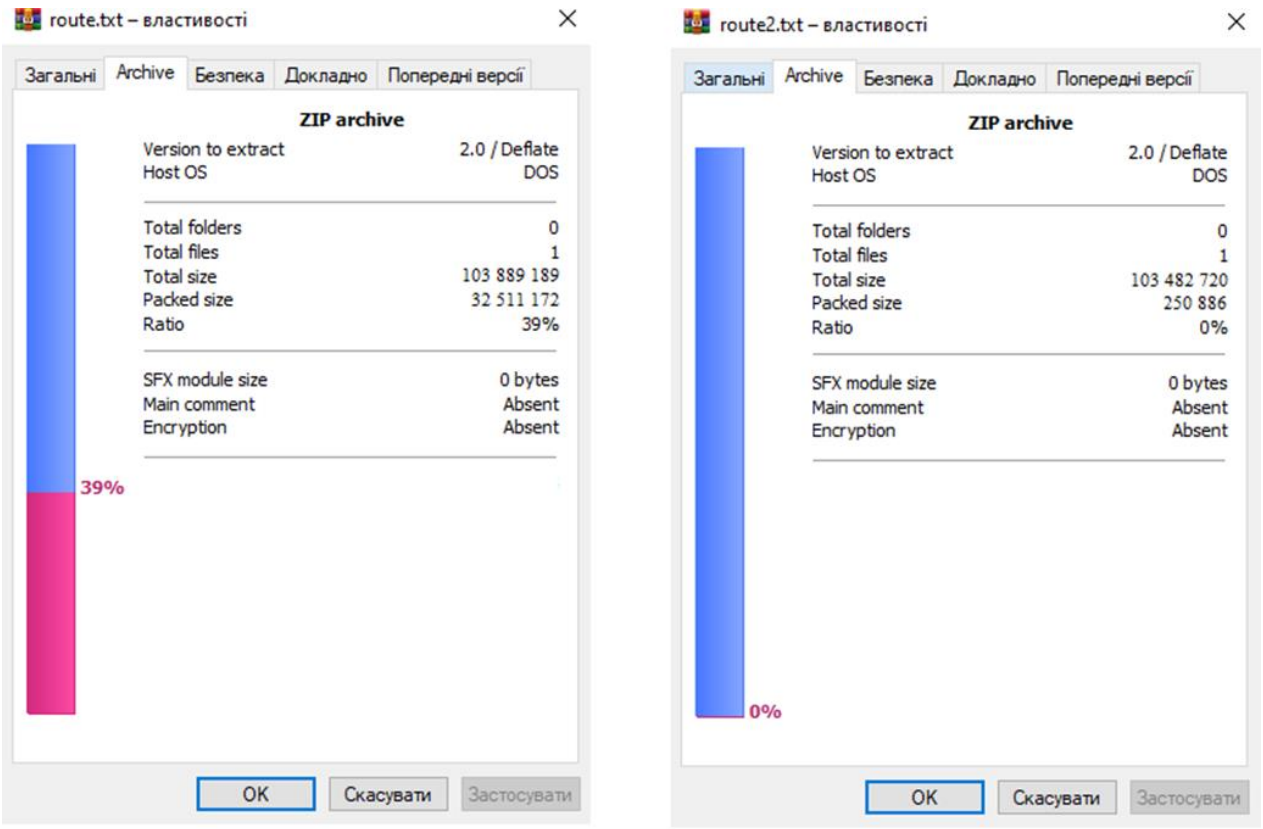


Рисунок 1.4 - Приклади маршрутної інформації: а) - змодельована інформації; б) - проста копії координат

Примітно, що розглянуте перетворення інформації являє собою стиснення без втрат (іншими словами, як є). Це пов'язано з тим, що в результаті зворотної обробки вихідна інформація отримується в точно такому ж вигляді, як і початкова.



а) б)

Рисунок 1.5 - Приклад стиснення маршрутної інформації шляхом простого архівування: а) - для модельованої інформації; б) - проста копії координат

1.4 Аналіз методів мінімізації обсягу інформації в залежності від швидкості об'єкта

У попередньому підрозділі обговорювалися можливі способи зменшення обсягу інформації про шлях без залежності від часових характеристик "подорожі". Передбачається, що координати положення об'єкта отримуються через однакові

проміжки часу незалежно від того, рухається об'єкт чи ні, або, в більш загальному випадку, яка його швидкість. Однак важливим аспектом цього процесу є те, що параметри зберігання даних залежать саме від швидкості. Насправді, якщо координати не змінюються взагалі (комбайн стоїть), то зберігати координати щосекунди чи навіть щохвилини явно не потрібно.

Тому необхідно ввести певну залежність від швидкості руху об'єкта в час оновлення координат Δt. Це, по суті, швидкість, з якою змінюються координати об'єкта. Тому зберігати координати слід лише в тому випадку, якщо вони змінилися достатньо сильно. Різні швидкості руху можуть призвести до різних фаз зменшення інформації:

– безпосередньо під час запису;
– при збереженні інформації про маршрут в архіві.

Обидва підходи мають свої переваги та недоліки. У той час як стиснення використовує більш-менш інтелектуальні алгоритми і вимагає значних обчислень, реалізація на рухомому пристрої (безпосередньо на комбайні) можлива лише за умови, що відповідна електронна база даних є значно складнішою і, відповідно, дорожчою. Позитивним моментом є те, що близько половини дискового простору (флеш-пам'яті) потрібно при зберіганні даних на рухомому об'єкті (тобто до того, як вони будуть внесені до центрального архіву). У цьому випадку економія пам'яті становитиме менше 50 відсотків. Таким чином, якщо параметри маршрутної інформації безпосередньо залежать від швидкості руху об'єкта (ще до занесення в архів), економія досягається за рахунок використання накопичувача вдвічі меншого розміру, але чим вища вартість, тим більш повнофункціональний обчислювальний пристрій (наприклад, процесори загального призначення замість мікроконтролерів) потрібно встановлювати. Враховуючи, що для встановлення звичайного процесора потрібне все обладнання (материнська плата, оперативна пам'ять, карти розширення), додаткові витрати на таке рішення значно перевищують економію.

Тому, безумовно, бажано виконувати інтелектуальну обробку даних на стороні сервера безпосередньо перед зберіганням даних в архіві. У випадку з мобільними об'єктами інформація буде зберігатися в нестиснутому вигляді, але це

зовсім не важливо з огляду на велику ємність сучасних флеш-карт, які підходять для тимчасового локального зберігання інформації.

Так, вартість найпростішої SD-карти об'ємом 16 Гбайт на 2023 рок становила близько 4 дол, а 32 Гб версії – біля 7 дол. Хоча відносна різниця може здатися великою (майже вдвічі), за абсолютною величиною різниця в 3 дол. абсолютно незначна, а до запису на невелику локальну флеш-пам'ять у самому комбайні Це на порядок (тобто до 100 разів) дешевше за вартість додаткового обладнання, що має бути встановлене на кожному комбайні для інтелектуального опрацювання (стиснення) даних. Тож, враховуючи низьку вартість флеш-пам'яті, достатньо записати поточні координати у флеш-пам'ять за допомогою недорогого мікроконтролера, завантажити їх на центральний сервер, використати обчислювальні ресурси центрального сервера для виконання стискання і потім записати результати на файловий архів та записати результати у файловий архів.

Таким чином, навіть якби координати записувалися щомиті, об'єм даних, отриманих за день, був би не більше ніж у 2 рази більшим (1.6), тобто близько 3,3 МБ; флеш-карта місткістю 16 ГБ забезпечить безперервний запис протягом 5000 днів, або близько 14 років. Знову ж таки, під час перенесення цієї інформації в архів, що містить сотні (або навіть тисячі) комбайнів даних, питання максимального стиснення стає принциповим і має вирішуватися якомога ефективніше.

Повертаючись до питання про залежність від швидкості координат, можна сказати, що в кінцевому результаті вона реалізується непрямим чином.

1.5 Методи підвищення надійності зберігання та захисту даних

Питання, пов'язані з надійністю та безпекою, традиційно є частиною будь-якого технічного проєкту, зокрема й проєкту з розроблення програмного забезпечення. У цьому розділі ці два питання буде розглянуто окремо.

Надійність будь-якої системи визначається як її здатність виконувати достатню кількість функцій для користувача протягом заданого періоду часу. Функціональність також розглядається тоді, коли система здатна працювати в повному обсязі, тобто без збоїв. Часткова втрата функціональності (або зниження якості послуг, що надаються системою) також часто береться до уваги. Іншими словами, виходить з ладу некритична підсистема (компонент), в результаті чого вся система продовжує функціонувати, але не в повному обсязі. Завданням інженера-проектувальника при аналізі надійності є ідентифікація критичних типів відмов проектованої системи та визначення їх числових характеристик.

У добре спроектованій системі зберігання маршрутної інформації необхідно враховувати два типи відмов:

– вихід з ладу флеш-пам'яті є найсмертельнішим, оскільки повністю знищує всі вже зібрані кореневі дані і унеможливлює подальше функціонування всієї системи;

– якщо критичний компонент системи вийде з ладу, подальша робота буде неможлива, але дані, які вже є на флеш-карті, не будуть знищені.

Тому далі ми розглянемо ці два сценарії відмови, які мають розкрити суть надійності.

У зв'язку з вищесказаним поняття надійності тісно пов'язане з імовірністю Q(t) того, що система відмовить протягом певного інтервалу часу t. Надійність визначається як імовірність взаємодоповнюючих подій:

$$P(t) = 1 – Q(t). \quad (1.12)$$

Іншими словами, надійність P(t) - це ймовірність того, що система перебуватиме в працездатному стані протягом часу t від початку експлуатації.

У разі складних систем, що складаються з декількох (або багатьох) компонентів (наприклад, систем запису і зберігання маршрутної інформації), розробляються схеми надійності, на основі яких надійність окремих компонентів визначає загальну інтегральну надійність.

Таким чином, проектована система містить у собі такі основні компоненти:

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

Набір критичних пристроїв може бути задано таким чином:

$$\Omega = \{\omega_1 = \text{“флеш”}, \omega_2 = \text{“контролер”}, \omega_3 = \text{“плата”}, \omega_4 = \text{“GPS”}\} \quad (1.13)$$

Якщо хоча б один із цих пристроїв виходить із ладу, то процес модифікації маршрутної інформації не може бути продовжено (хоча якщо компонент, що відмовив, не є мікросхемою флеш-пам'яті, то маршрутна інформація, що була доступною до відмови, може бути переписана на центральний сервер). Ця умова означає, що на схема надійності (рис. 1.6) усі ці елементи з'єднано послідовно.

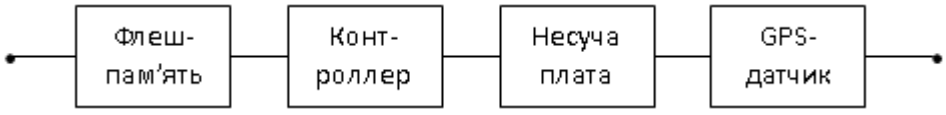


Рисунок 1.6 - Схема надійності системи зберігання маршрутних даних.

За дотримання схеми, наведеної на рис. 1.6, і загальних правил побудови функцій надійності результуюча надійність дорівнює добутку надійностей окремих елементів:

$$P(t) = P_{ф}(t) \cdot P_{мк}(t) \cdot P_{пл}(t) \cdot P_{GPS}(t) = \prod_{i=1}^{4} P_i(t), \quad (1.14)$$

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, що кожен коефіцієнт надійності в (1.14) можна оцінити за загальноприйнятим експоненціальним законом:

$$Pi(t) = e^{-\lambda it}, \tag{1.15}$$

Де λ_i - інтенсивність відмов і-го компонента, що є одним з елементів (1.12). Як відомо, ця величина легко розраховується на основі параметра MTBF (Mean Time Before Failure):

$$\lambda_i = \frac{1}{T_i}, \tag{1.16}$$

де T_i - середній час інтервалу несправності і-го елемента множини (1.13).

Тоді, з урахуванням (1.16) і (1.15), рівняння (1.14) набуває вигляду:

$$P(t) = \prod_{i=1}^{4} P_i(t) = \prod_{i=1}^{4} e^{-\lambda_i t} = \prod_{i=1}^{4} e^{-\frac{t}{T_i}} = \exp\left(-t \sum_{i=1}^{4} \frac{1}{T_i} \right) \tag{1.17}$$

Це рівняння (1.17) може бути використане для розрахунку надійності складних систем, що складаються з компонентів (1.13), значення наробітку на відмову яких часто наводять у довідкових даних, що потребують лише уточнення конкретної моделі відповідного обладнання.

Якщо припустити, що така відмова - це втрата всіх даних висхідного тракту, то ланцюжок на (1.17) вироджується до одного осередку (надійність флеш-пам'яті). Таким чином, надійність усієї системи зберігання даних дорівнює надійності цього елемента:

$$P(t) = P_{\phi}(t) \tag{1.18}$$

Таким чином, в рамках стандартної теорії надійності вирішується проблема забезпечення відмовостійкості системи, що розробляється. Що стосується інформаційної безпеки, то добре відомо, що всі заходи безпеки можна розділити на три типи, залежно від типу загрози, яку вони становлять:

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

У цьому випадку, само собою зрозуміло, що найбільшу увагу слід приділяти засобам забезпечення цілісності маршрутної інформації. Це пов'язано з тим, що, на жаль, люди, які тривалий час перебувають у безпосередній близькості до критичної інформації (оператори сільськогосподарської техніки), становлять найбільшу загрозу для цієї інформації (наприклад, вони можуть видалити, знищити або пошкодити носії).

Загрози конфіденційності інформації про маршрутизацію майже не існують і їх неможливо ідентифікувати; те ж саме стосується і загроз доступності (гніздо, в яке вставляється флеш-пам'ять, тимчасово пошкоджено, і її вміст навряд чи буде збережено). Тому основна увага в інформаційній безпеці приділяється забезпеченню цілісності інформації.

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

- юридична;
- організаційн;
- технічні.

Юридичні засоби не можуть вирішити проблему забезпечення цілісності. Це пов'язано з тим, що чітко визначено, що будь-якій особі (переважно співробітникам компанії) заборонено руйнувати обладнання або пошкоджувати окремі компоненти. Технічні заходи також не підходять для розв'язання проблеми через їхню високу вартість (оскільки практично всі технічні рішення вимагають розроблення проєкту, створення прототипу, тестування, впровадження та експлуатаційних витрат). Тому найкращим рішенням у цьому випадку є реалізація організаційних заходів. Зокрема, мають бути реалізовані такі заходи:

– пристрої, що зберігають маршрутну інформацію (мікроконтролер, GPS-модуль, флеш-пам'ять), встановлюються окремо від інших засобів керування електронними компонентами рухомої техніки;

– пристрій повинен бути захищений надійним корпусом, бажано звичайною металевою пластиною;

– забезпечення подачи електроживлення на пристрій (на початковому етапі для цього можна вибрати місце, розташоване якомога ближче до системи електропостачання);

– вилучення і перенесення заповнених флеш-карт у ЦОД, а також доставка порожніх карт і їхнє встановлення мають здійснюватися спеціально виділеними співробітниками;

– якщо пристрій оснащений GSM-модулем, процедура зчитування флеш-карти повинна бути повністю автоматичною і прозорою для оператора;

– за жодних обставин оператор не повинен мати доступ до флеш-карти або блоку керування пристроєм пошуку маршрутної інформації.

Дотримуючись простих принципів, описаних вище, можна досягти належного рівня надійності та захисту даних у системах зберігання маршрутної інформації.

1.6 Висновки до розділу 1

У цьому розділі розглядається основна інформація про шляхи та засоби вирішення проблеми зберігання маршрутної інформації, які наразі існують у відповідних галузях. По-перше, проаналізовано варіанти використання маршрутної інформації в різних системах, в тому числі в сільськогосподарській техніці. Процес генерації маршрутних даних, необхідних для реалістичної оцінки ступеня стисливості маршрутної інформації, змодельовано в середовищі Mathcad.

Результати показують, що інформація в її непідготовленому прямому вигляді може бути стиснута приблизно до 30% від її початкового значення.

У цьому розділі пояснюється, як отримується маршрутна інформація, аналізується її структура та властивості, а також описуються можливі додаткові методи додаткового стиснення. Також описано залежність параметрів маршрутної інформації від швидкості, надійність та захист інформації.

Загалом було встановлено, що існують способи поліпшення стиснення маршрутної інформації.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Обґрунтування вибору методу мінімізації маршрутних даних

До розглянутої системи можуть бути застосовані найрізноманітніші методи мінімізації маршрутних даних. Розглянемо характеристики основних із них.

По-перше, можна використовувати архівування - поширений метод оборотного стиснення інформації. Аналіз його характеристик і шляхів удосконалення методів архівації виходить далеко за рамки цієї роботи і являє собою великий пласт знань у галузі теорії інформації. Тому в цьому дослідженні архівування використовується в крайньому випадку, коли всі інші методи стиснення вже використовуються відповідно до стандартних методик (наприклад, алгоритм ZIP).

Другий можливий підхід полягає в урахуванні фізичної природи стискуваних даних. Тобто практично всі числа у файлі маршрутної інформації (за винятком деяких службових даних), являють собою пари координат точки на земній поверхні з координатами сусідніх точок. Полілінія формується з окремих точок, розташованих послідовно, і за достатньої щільності вузлів може бути досить точно представлена кривою або сплайном. Завдання полягає в тому, щоб за набором послідовних точок знайти коефіцієнти сплайна відповідного ступіня. При цьому

набір коефіцієнтів повинен займати менше байт, ніж набір координат точок, що утворюють сплайн.

Нагадаємо, що координати двох точок лежать точно на прямій, рівняння якої має два коефіцієнти (y = ax + b, два коефіцієнти a і b); три точки лежать точно на квадратичній кривій, тобто параболі з трьома коефіцієнтами (y = ax2 + bx + c, три коефіцієнти a, b, c).

Для того щоб розташувати рівно n точок на кривій ступеня n, потрібно рівно n коефіцієнтів многочлена ступеня n:

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

Тому під час використання поліноміального рівняння порядку n необхідно зберігати тільки координати xi, а всі значення yi (всього їх n) обчислюються за рівнянням (2.1), тому їх зберігати не потрібно. Однак, крім усіх x, необхідно зберігати і всі коефіцієнти ak у рівнянні (2.1), що також становить у сумі n. Тому якщо точність коефіцієнтів ak не нижча за точність коефіцієнтів xi, то ніякої економії від такої заміни не видно. В іншому разі (тобто за умови зниження точності зберігання коефіцієнтів ak), з огляду на те, що значення ui обчислюється арифметично за цими коефіцієнтами, значення цих результатів визначатиметься неточно (оскільки коефіцієнти в (2.1) є ступінчастими, то неточність визначення розрядів зберігається: наприклад, якщо коефіцієнт ak має похибку до другого десяткового знака, то й сумарний результат, тобто ui, матиме таку ж саму похибку до другого десяткового знака). З огляду на загальну кількість коефіцієнтів ak і можливість неточностей досить великого порядку в кожному з них, сумарна похибка у визначенні координат ui згідно з (2.1) може спотворити маршрут і зробити абсолютно неможливим його використання (перегляд, перевірка тощо).

Тому перехід до сплайнових апроксимацій у завданні збереження маршрутної інформації недоцільний. Розглянемо інший, більш універсальний метод перетворення інформації, який може слугувати цілям цього дослідження.

Як відомо, крім безлічі дійсних чисел, які можна використовувати для вимірювання фізичних величин, існує цілий клас комплексних чисел, що включає спеціальні одиниці вимірювання комплексних чисел $i = \sqrt{-1}$. Оскільки цей символ є уявним числом, комплексні числа не можуть бути використані для опису реальних явищ. Однак такі числа широко використовуються при перетворенні сигналів і, в ширшому сенсі, в інформаційних проблемах.

Суть низки "комплексних" перетворень (їх називають перетвореннями Фур'є, хоча історично назва "перетворення Фур'є" вперше була використана для перетворень, що ґрунтуються на синусі та косинусі тригонометричних функцій) ґрунтується на двох правилах:

– як із множини дійсних чисел отримати відповідну множину комплексних чисел;

– як перейти від множини комплексних чисел назад до вихідної множини дійсних чисел.

Доцільність застосування перетворень Фур'є зумовлена тим, що те, що дуже важко зробити з дійсними числами (наприклад, розв'язати диференціальні рівняння), набагато простіше зробити в їхньому комплексному числовому образі (диференціальні рівняння можна замінити алгебраїчними рівняннями, які завжди набагато простіше розв'язувати). Таким чином, корисна схема застосування інтегральних перетворень типу Фур'є має такий вигляд:

а) висхідний реальний сигнал f1(t) (інформація) піддається комплексному перетворенню, тобто оригінал комплексно перетворюється для отримання відповідного комплексного виразу F1(p), або зображення:

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

де K(p, t) - ядро інтегрального перетворення;

T - інтегральна область реальної області, яка використовується в цьому ядрі K(p, t).

б) для комплексного зображення $F_1(p)$ перетвореного вхідного сигналу виконати необхідні дії, спрямовані на розв'язання першої задачі обробки вихідного вхідного сигналу для отримання модифікованого комплексного зображення $F_2(p)$;

в) виконується зворотне комплексне перетворення, і за модифікованим комплексним зображенням $F_2(p)$ відтворюється бажаний вихідний сигнал $f_2(t)$ у звичайному форматі дійсних чисел:

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

Таким чином, шлях виконання операцій:

$$f_1(t) \rightarrow F_1(p) \rightarrow F_2(p) \rightarrow f_2(t) \tag{2.4}$$

у разі використання інтегральних перетворень типу Фур'є це набагато простіше, ніж безпосереднє опрацювання інформації $f_1(t) \rightarrow f_2(t)$.

Теоретично схема (2.4) придатна для побудови системи стиснення маршрутної інформації, але тільки за виконання таких умов:

а) якщо перетворений сигнал $F_1(p)$ може бути якимось чином зменшений в об'ємі, що призводить до отримання модифікованого експресивного зображення $F_2(p)$;

б) якщо розмір F2(p) у пам'яті комп'ютера менший за розмір наростаючого речового сигналу f1(t). Іншими словами, вона має бути виконана:

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

де оператор sizeof() вказує на те, що розмір береться в байтах. Якщо (2.5) не виконується, то ніякого стиснення не відбувається і все перетворення просто не має сенсу;

в) $F_1(p)$ і $F_2(p)$ є різними зображеннями і, отже, у загальному випадку відрізняються від вихідного сигналу $f_1(t)$. Проте ці два сигнали мають бути близькими один до одного. Іншими словами:

$$diff\,(f_1(t), f_2(t)) < \Delta f, \tag{2.6}$$

де оператор diff являє собою операцію знаходження різниці між двома сигналами, а Δf - пороговий допуск на цю різницю. Як конкретну операцію знаходження різниці можна взяти інтеграл від арифметичної різниці в області визначення двох функцій:

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

Для того щоб визначити доцільність використання цієї схеми стиснення в реальній системі зберігання маршрутної інформації, необхідно проаналізувати її потенційну ефективність. Для цього визначимо стандартне перетворення Фур'є, ядро якого має вигляд K(p, t) = e-pt, і реалізуємо інтегральне перетворення в уже використаній вище системі Mathcad.

У програмному середовищі є функція fft(), що виконує "швидке" перетворення Фур'є, особливістю якої є те, що вхідний вектор чисел має містити 2m елементів (тобто їхня кількість має бути кратна 2). З огляду на те, що потік координат маршрутної інформації безперервний, розбиття його на 2m числових частин не є особливою проблемою: на виході функції fft() отримують 2m-1 комплексних чисел однакової точності, до того ж кожне комплексне число є парою двох дійсних чисел, тому дане перетворення ($f_1(t) \rightarrow F_1(p)$) не змінює загального обсягу інформації при перетворенні. Водночас зображення $F_1(p)$ можна стиснути, виконавши перетворення $F_1(p) \rightarrow F_2(p)$. Наприклад, для цього достатньо встановити в нуль оцінний модуль кожного комплексу і число, модуль якого менший за поріг ΔF (тобто число, внесок якого в загальний результат дуже малий)

(тим самим зменшивши інформативність $F_2(p)$). Після цієї процедури стиснене зображення $F_2(p)$ може бути піддано зворотному перетворенню Фур'є для отримання нового оригіналу $f_2(t)$. Завдання полягає в тому, щоб оцінити, як впливає вибір порога ΔF на різницю (2.4) між двома реальними сигналами, сформованими на заданому ΔF.

Для вивчення зазначених питань було реалізовано прості користувацькі функції системи MathCad:

– функція compressZero(fCoords, ΔF) обертає в нуль усі комплексні числа в матриці fCoords, модуль яких менший за заданий поріг ΔF.

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

– функція NumZeroed(fCoords, ΔF) підраховує кількість комплексних чисел у матриці fCoords, модуль яких менший за заданий поріг ΔF.

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

Результати застосування цих функцій (приклади застосування наведено на рис. 2.1) і для різних порогових значень ΔF зведено в табл. 2.1.

Таблиця 2.1. Параметри системи стиснення маршрутних даних відповідно до порога ΔF

№	ΔF	Обсяг фрейму, відліків	Занулених фреймів	На скільки зменшуєтьс я обсяг, %	Середнє відхилення, diff, градусів	Середнє відхилення, метрів
1.	$0{,}5\cdot10^{-2}$	32768	460	1,4	0,0005	50

Продовження таблиці 2.1

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

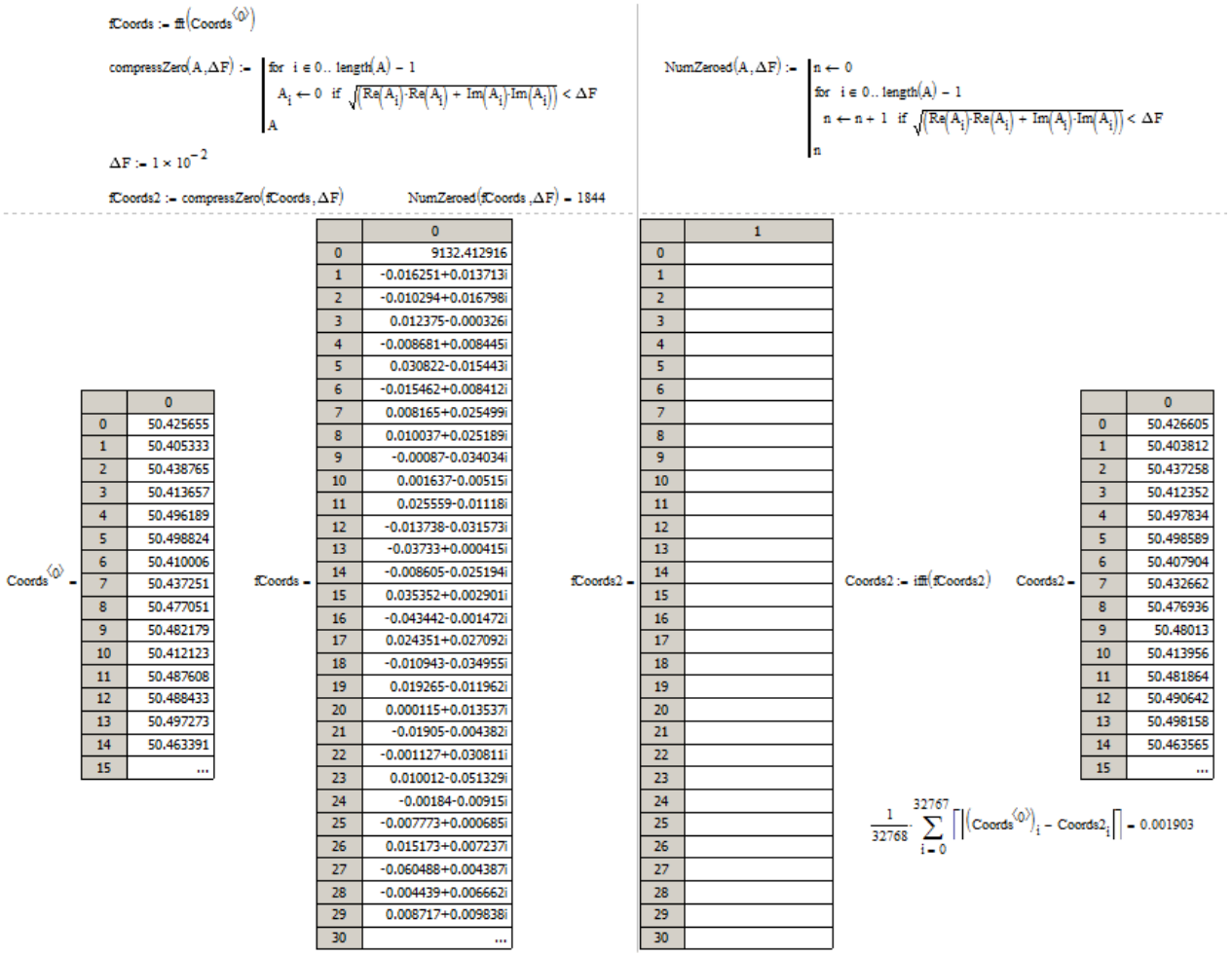


Рисунок 2.1 - Приклад стиснення маршрутної інформації за допомогою інтегрального перетворення (з порогом ΔF = 10-2)

Аналіз, наведений у табл. 2.1, показує, що за умов, які задовольняють (2.6)-(2.7), тобто забезпечують достатню якість відновленого сигналу, застосувати

ефективне стиснення на основі інтегрального перетворення за схемою (2.4), не є можливим.

Дійсно, перший рядок показує поріг ΔF = 0,5·10-2, параметр 460 комплексних чисел відкинуто, тобто обчислювальна деформація становить близько 1,4%. Середнє відхилення однієї точки при зворотному перетворенні, тобто при відтворенні $f_2(t)$, становить близько 50 м, що перебуває якраз на межі точності маршрутної інформації (в умовах пересіченій місцевості цю межу дещо перевищено, оскільки випадкове відхилення в 50 м може повністю спотворити зміст маршрутної інформації). Це означає, що середнє відхилення записаних точок маршруту, що виникає в результаті стиснення, спричиненого інтегральним перетворенням, має бути зменшене, тобто поріг ΔF має бути знижений. Однак навіть розглянуте значення ΔF=0,5·10-2 зменшує обсяг інформації лише приблизно на 1%, а менший поріг ще більше зменшить обсяг інформації, що відкидається, і становитиме менш ніж 1%, що абсолютно неефективно як завдання стиснення даних.

Таким чином, ані сплайн-підхід, ані методи інтегрального перетворення Фур'є не дають ефективних результатів для задачі стиснення маршрутної інформації перед її збереженням в архіві. Водночас, враховуючи інформацію, наведену в першому розділі, можна зробити висновок, що найбільш доцільним способом обробки маршрутної інформації є розгляд її характеристик і стиснення на основі цього аналізу, без необхідності застосування складних математичних операцій.

2.2 Аналіз можливостей поліпшення методів

Тому для проведення мінімізації використовується відносно простий з математичного погляду метод, заснований на аналізі характеристик маршрутної інформації.

По-перше, слід зазначити, що значення координат, які надаються GPS-датчиками, записуються в частках градуса і містять одну мільйонну частку градуса (див., наприклад, (1.2). Розрахунки в (1.7) показують, що одна мільйонна частка градуса відповідає відстані 0,1 м, що зовсім не потрібно в цільовій зоні (занадто велика точність). Це означає, що від однієї мільйонної частки градуса можна відмовитися без втрати важливої та значущої інформації й обмежитися 1/100 000 градуса, що дає відстань між двома послідовними позиціями близько одного метра. Зменшення кількості бітів, що відображаються в обох координатах, таким чином, призведе до зменшення обсягу інформації, що зберігається, приблизно на $1/8 \approx 12\%$, що не так вже й мало. Видалення додаткових бітів уже не виправдане, оскільки це призводить до помилок під час визначення положення в метрах відповідно до (1.8).

Якщо подивитися на координати з іншого боку (з верхніх цифр), то можна побачити інший тип повторення, де повторюється саме значення градуса, а також десяткова крапка (наприклад, на рисунку 1.4 всі записи довготи починаються з 30.5, а всі записи широти - з 50.4). Очевидно, що наявність таких повторів також може бути ефективно використана для стиснення маршрутної інформації. Наприклад, тег BASE (або скорочено B) можна використовувати для встановлення базових значень довготи та широти, до яких можна додавати закріплені значення в наступному рядку; наступний тег типу BASE з'являється, коли змінюється частина координат, вказана в цьому тезі.

Наприклад, при роботі в межах сільськогосподарської території різниця в кутовому переміщенні між довготою і широтою становить 0,1º. Це тому, що, згідно з (1.10), така кутова відстань відповідає лінійному переміщенню близько 10 км. Навіть якщо зміна кутових координат знаходиться в межах 0,1 градуса, це трапляється вкрай рідко, і цього можна уникнути, ввівши тег типу BASE.

Розгляд двох основних тегів окремо дає змогу застосовувати більш гнучкий підхід:

- B_X - зміна базової частини довготи;
- B_Y - зміна базової частини широти.

У цьому випадку фрагмент маршрутної інформації матиме вигляд, як показано на рисунку 2.2, б. Для порівняння, на рисунку 2.2, а показано фрагмент того ж файлу до того, як було відкинуто зайву мільйонну частку степеня і закодовано основну частину координат.

Таке перетворення зменшує обсяг інформації приблизно вдвічі, а оскільки відмова в одну мільйонну частку степеня є несуттєвою для цільової області, то втрат практично немає.

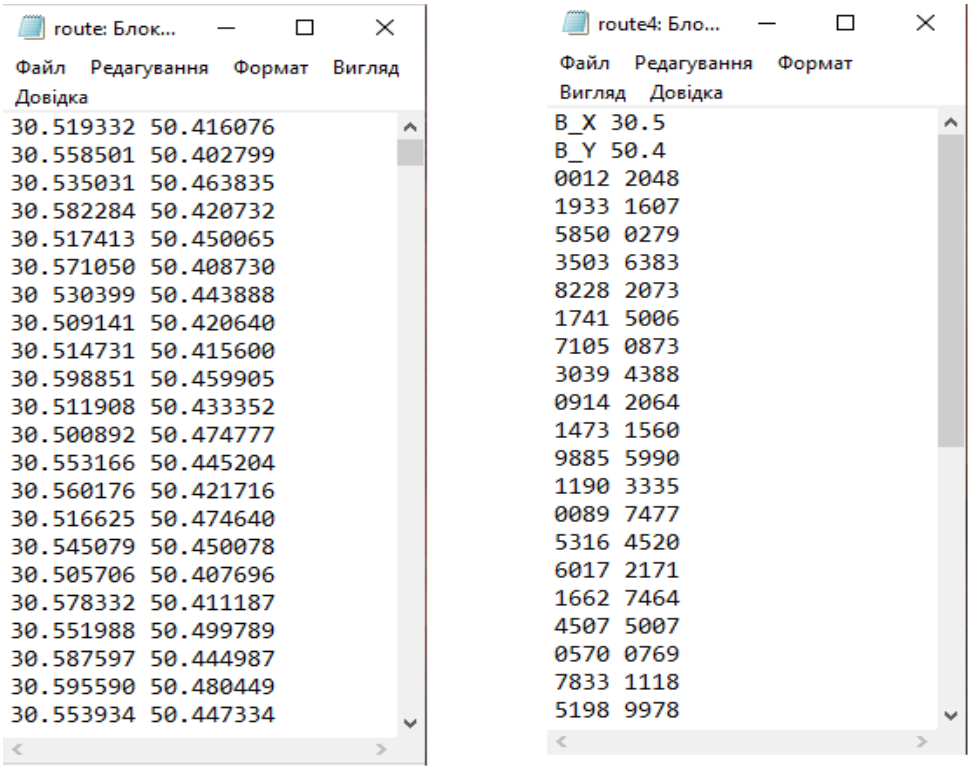


Рисунок 2.2. - Перетворення координатної інформації: а) - оригінал, б) - та сама інформація після введення базової частини кожної координати, відкинувши мільйонні частки.

На додаток до описаних вище дій наступний крок стиснення може бути реалізовано шляхом перетворення кожної з чотиризначних координат (рис. 2.2, б) у числовий формат: оскільки цифр лише чотири, кожне таке число поміщається у двобайтову змінну типу short int. Таким чином, замість того щоб у текстовому файлі для зберігання чотиризначного "слова" використовувати чотири байти, у кінцевому стислому форматі кожна координата займає два байти, що майже вдвічі скорочує

загальний обсяг інформації (без урахування службової інформації, такої як BASE-мітки).

Тут слід зазначити, чому саме десяткові частини координат включаються до бази даних (окрім самих координат, звичайно), а соті, тисячні, десятитисячні та стотисячні долі залишаються для формування заданого чотиризначного числа. Проблема вибору бітів для включення в базу даних є класичною оптимізаційною задачею з суперечливими вимогами.

По-перше, службові записи, такі як слово BASE (або B_X і B_Y), слід використовувати якомога рідше, оскільки їх велика кількість збільшує розмір файлу. Тому слід включати якомога менше бітів, наприклад, самі координати не повинні містити дробових частин. Однак, для кожного запису положення об'єкта необхідно зберігати до п'яти цифр, тобто максимальне число - 99999, що вимагає більше двох байт пам'яті комп'ютера. Оскільки створити 3-байтну змінну стандартним способом неможливо (технічно дуже складно використовувати таку структуру/регістр), то фактично для таких чисел необхідно використовувати 4-байтну змінну типу int. Таким чином, зберігаючи тег BASE, ми більше втрачаємо у представленні кожної координати. Включення десяткових бітів у базу на додаток до цілих значень дещо збільшує частоту використання тегів BASE, але залишає по 4 біти для кожної координати, тобто максимальне число 9999, яке можна зберігати у 2-байтній змінній типу short int. Таким чином досягається найбільша економія дискового простору при зберіганні всього файлу: При подальшому збільшенні кількості бітів, що включаються в BASE, мітки BASE будуть з'являтися ще частіше при включенні однієї сотої степеня, але все одно доведеться виділяти по два байти для кожної координати (для зберігання чисел від 000 до 999, які не поміщаються в один байт в такій змінній). Тож цей варіант однозначно гірший за попередній. Останній варіант, який варто розглянути, - включити в базу даних частоти з точністю до тисячної частки градуса. У цьому випадку в одному байті для кожної координати можна зберігати лише два біти (тобто числа від 00 до 99). Однак, якщо потрібно внести зміну навіть на одну тисячну градуса, необхідно ввести окрему мітку типу BASE, де 0,001° відповідає лінійному переміщенню приблизно на 100 м

на земній поверхні. Тому рекомендується вводити 1/1000 градуса в мітці BASE, якщо сусідні точки маршруту знаходяться набагато ближче, ніж 100 метрів. Якщо сусідні точки знаходяться на відстані десятків метрів одна від одної (тобто тисячні частки градуса змінюються досить часто), також рекомендується обмежити значення BASE десятими частками градуса. Ця інформація узагальнена в таблиці 2.2. Ця таблиця показує середню лінійну відстань, на яку потрібно перемістити точку, щоб змінити її відповідне значення BASE наполовину, оскільки точку можна переміщати в обох напрямках.

Таблиця 2.2 - Аналіз ефективності розподілу координатних бітів між базами даних і окремими записами (у припущенні, що радіус Землі становить 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, якщо узагальнити питання розподілу координатних бітів між базою даних та окремими записами, то варіант 4 є оптимальним для запису маршрутної інформації для сільськогосподарської техніки.

Усі перераховані вище вдосконалення відносно незалежні й можуть бути застосовані в комбінації для формування відповідного алгоритму.

2.3 Розробка алгоритмів мінімізації маршрутних даних

Таким чином, у попередньому підрозділі було запропоновано різні операції для стиснення маршрутної інформації, що надається GPS-датчиком, і на основі цього було побудовано алгоритм, який включає наступні кроки:

– відкидання одну частину на мільйон (шостий знак після коми) у всіх доступних координатах;

– вибір кількісті знаків після коми, яку буде включено до бази даних (варіант 2 або 4 у табл. 2.2), і відповідно решту цифр, які необхідно залишити в кожній координаті (залежно від характеру маршрутної інформації; варіант 4 у табл. 2.2 є оптимальним для переміщення сільськогосподарської техніки по полю);

– сформувати мітку типу BASE, що задає базову (незмінну в межах великого блока даних) початкову частину кожної координати, і відкинути цю базову частину в межах одного блока за всіма доступними в блоці координатами;

– закодувати чотиризначне число, що відповідає одній із координат, у числову змінну типу short int що у кроці 2 було обрано варіант 2 у табл. 2.2;

– закодувати кожне двозначне число, що відповідає одній із координат, у числову змінну типу char/byte, на кроці 2 було обрано варіант 4 у табл. 2.2;

– архівування отриманого файлу разом з їхніми координатами за допомогою архіватора.

Схему цього алгоритму наведено на рисунке 2.3.

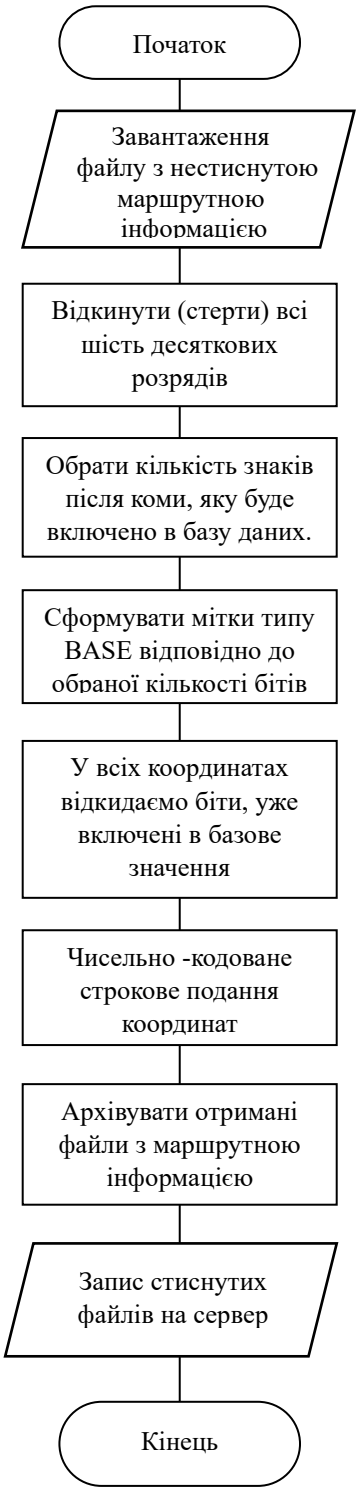


Рисунок 2.3 - Схема алгоритму стиснення маршрутної інформації

За необхідності спеціалізоване програмне забезпечення, що розробляється в рамках цієї роботи, має також давати змогу користувачам виконувати зворотне перетворення й отримувати інформацію про маршрут у зрозумілому (текстовому) форматі, який може бути розпізнаний такими системами, як Google Maps.

Для тестування цього алгоритму на вхід має подаватися реальна інформація або параметрично близька до реальної. У підрозділі 1.3 маршрутна інформація вже моделювалася, але представлений там рудиментарний метод генерації був цілком задовільним, оскільки його мета була зовсім іншою і полягала в оцінці ступеня стиснення загальним архіваріусом. Однак нині, у зв'язку з необхідністю візуалізації шляхів, описана раніше процедура генерації випадкових координат, коли наступна координата не пов'язана з попередньою, вже не є задовільною. Однак дуже легко розробити складну процедуру генерації випадкових маршрутних даних, параметри і зовнішній вигляд яких дуже близькі до реального руху комбайна на полі. Реалізація, як і раніше, створюється в Mathcad за допомогою користувацької функції GenerateWay(), яка приймає п'ять вхідних параметрів:

– широта центральної точки поля;

– довгота точки поля, що вважається центральною точкою;

– допустиме відхилення рухомого об'єкта від центральної точки поля, в одиницях градусів;

– коефіцієнт швидкості комбайна, у градусах/секунду;

– напрямок швидкості комбайна, що задається єдиним кутом α;

– кількість точок для генерації.

$$\text{GenerateWay}(x0, y0, \text{diam}, v, \text{alfa}, N) := \left| \begin{array}{l} X_{0,0} \leftarrow x0 \\ X_{0,1} \leftarrow y0 \\ \text{for } i \in 1..N-1 \\ \quad \left| \begin{array}{l} X_{i,0} \leftarrow X_{i-1,0} + v \cdot \cos(\text{alfa}) \\ X_{i,1} \leftarrow X_{i-1,1} + v \cdot \sin(\text{alfa}) \\ \text{alfa} \leftarrow \text{alfa} \cdot \left(0.9999 + \frac{\text{rnd}(2)}{10000}\right) \\ \text{alfa} \leftarrow \text{alfa} + \pi \text{ if } \sqrt{\left(X_{i,0} - x0\right)^2 + \left(X_{i,1} - y0\right)^2} > \text{diam} \end{array} \right. \\ X \end{array} \right.$$

$$NN := 20000$$

$$\text{Res} := \text{GenerateWay}\left(50.4, 30.5, 0.02, 2 \cdot 10^{-5}, \frac{\text{rnd}(314)}{100}, NN\right) \qquad \text{out} := \text{WRITEPRN}(\text{"route2.txt"}, \text{Res})$$

$$i := 0..NN$$

Рисунок 2.4 - Фрагмент кода розробленої функції

Результат роботи цієї функції при виклику GenerateWay(), (рис.2.5).

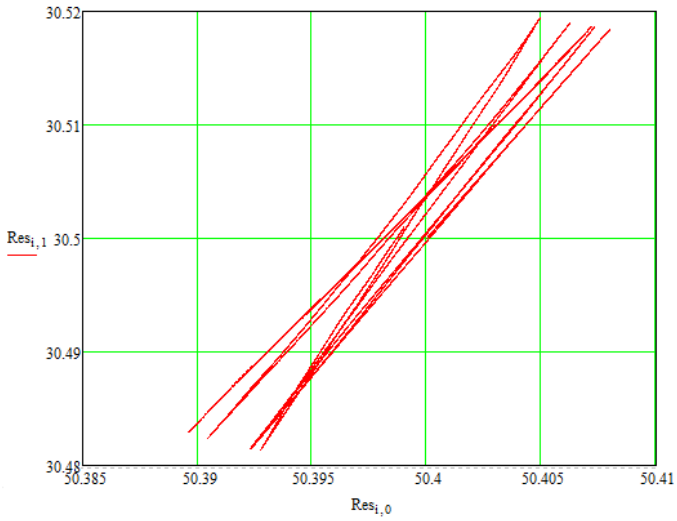


Рисунок 2.5 - Приклад формування модельних даних для маршруту руху сільськогосподарської техніки (у межах одного поля)

Відповідний "великий" набір координат записується у файл (route2.txt) і може бути використаний, зокрема, для реалізації методу стиснення за алгоритмом на рисунку 2.3.

2.4 Оцінка залежності методів та властивостей моделі для мінімізації кількості інформації про швидкість руху об'єкта

Як згадувалося в першому розділі, параметри інформації про орієнтацію часто залежать від швидкості транспортного засобу, що рухається. Зокрема, найбільш очевидним вираженням цієї залежності є те, що коли швидкість дорівнює нулю (тобто транспортний засіб простоює), координати просто повторюються протягом певного періоду часу. Звичайно, немає сенсу зберігати дані, які повторюються багато разів поспіль, і найочевиднішим підходом у цьому випадку є

введення у файл кількості повторень кожного набору координат, виключаючи при цьому дублікати координат. Це можна зробити за допомогою спеціальних міток або тегів (доступні додатково до BASE).

Окрім введення тегу BASE (B_X, B_Y), у файл також потрібно ввести тег REP (від REPetition - повторення). Це означає, що остання координата повторюється N разів і є дуже ефективним і необхідним способом стиснення даних маршруту, коли транспортний засіб простоює в дорозі.

Зверніть увагу, що немає сенсу застосовувати тег REP з двох частин (R_X, R_Y і т.д.): якщо об'єкт не рухається, обидві координати не будуть змінюватися одночасно; якщо він рухається, обидві координати будуть змінюватися, але, ймовірно, тільки найменший біт координат з кожного боку).

Таким чином, файли, що містять координати, конвертуються за прикладом, показаним на рисунку 2.6. Зрозуміло, що у випадках, коли тривалість відключення велика або є багато коротких відключень, такий підхід може значно заощадити місце, необхідне для зберігання маршрутної інформації.

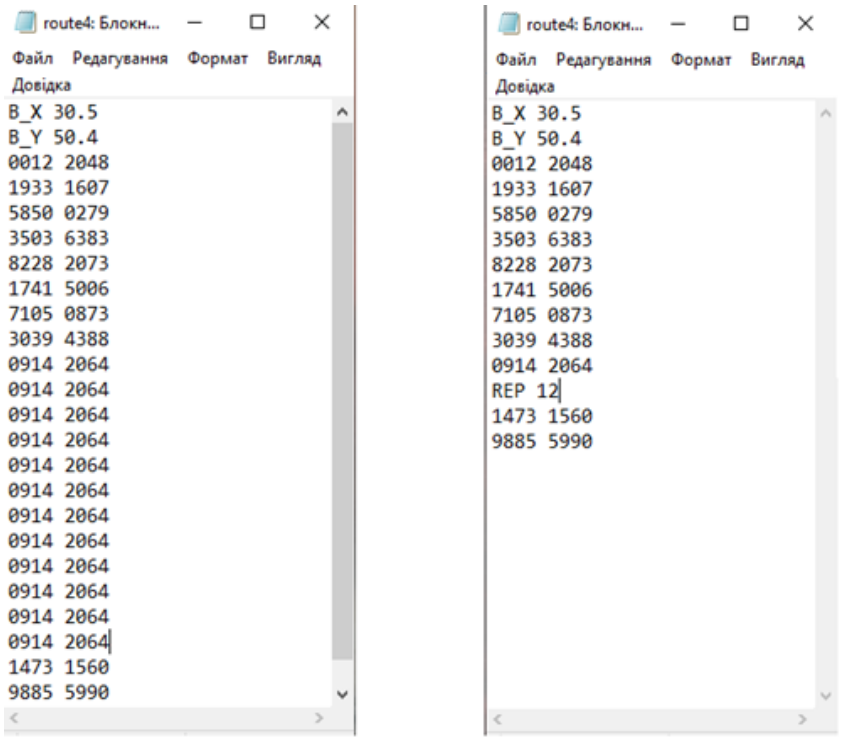


Рисунок 2.6 – Файли, з кореневою інформацією: а) - до стиснення, б) - після застосування мітки REP

При введенні описаних тегів REP має бути реалізовано алгоритм і відповідну послідовність операцій, наведені на рис. 2.3. Алгоритм 2.3 і відповідна послідовність операцій. Таким чином, отримано нову алгоритмічну схему стиснення маршрутної інформації (рис. 2.7) і підсумкову послідовність операцій:

– відкидання одну частину на мільйон (шостий знак після коми) у всіх доступних координатах;

– вибір кількісті знаків після коми, яку буде включено до бази даних (варіант 2 або 4 у табл. 2.2), і відповідно решту цифр, які необхідно залишити в кожній координаті (залежно від характеру маршрутної інформації; варіант 4 у табл. 2.2 є оптимальним для переміщення сільськогосподарської техніки по полю);

– створіть тег типу BASE, який визначає базову (незмінну в межах великого блоку даних) початкову частину кожної координати і призначає цю базову частину всім координатам у блоці в межах одного блоку;

– створювання міток типу REP і відкидання повторюваних координат, якщо принаймні дві або більше послідовних точок мають однакові координати (якщо обладнання простоює або зупинене);

– закодувати чотиризначне число, що відповідає одній з координат, в одну числову змінну типу short int якщо, у кроці 2 було обрано варіант 2 у табл. 2.2;

– закодувати двозначне число, що відповідає одній із координат, у числову змінну типу char/byte якщо, у кроці 2 було обрано варіант 2 у табл. 2.2;

– архівування отриманого файлу разом з їхніми координатами за допомогою архіватора.

Тому в цьому підрозділі розглядається варіант введення в систему залежно від швидкості об'єкта параметра стиснення інформації про шлях (пропонується спеціальна мітка REP, скорочено R) на прикладі стану зупинки або простою, коли координати, що повторюються, відкидаються, і залишається тільки перше входження із зазначенням числа повторень.

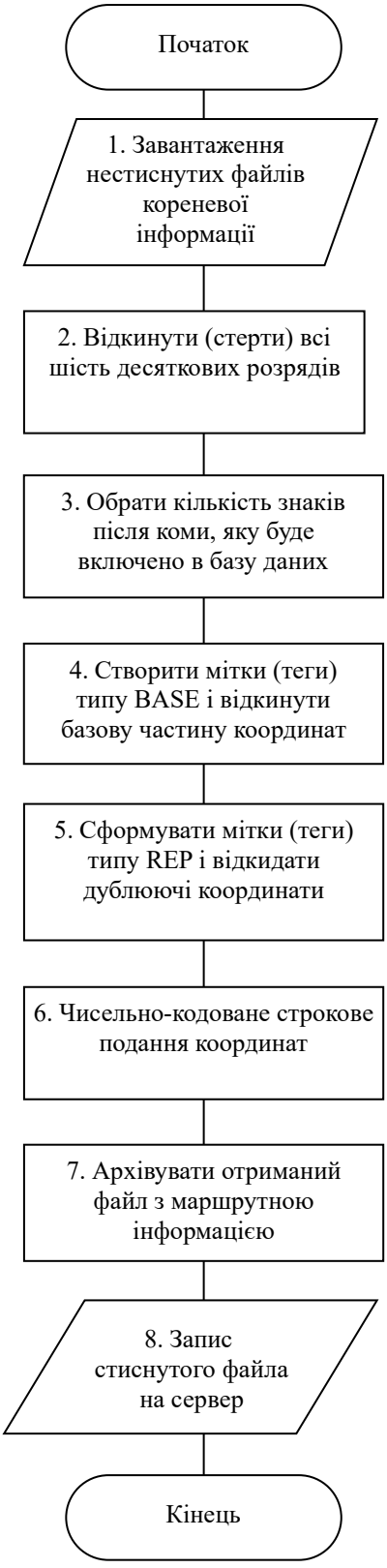


Рисунок 2.7 – Версія алгоритму стиснення маршрутної інформації (з урахуванням повторення окремих координат)

2.5 Висновки до розділу 2

Тому в цьому розділі пропонується метод стиснення маршрутної інформації відповідно до таких принципів:

– відмова від мінімальних (частки на мільйон) ступенів, що відповідають зміщенням поверхні землі, близьким до 0,1 м (згідно з (1.7)) - оскільки їх використання в розглянутій системі не потрібне;

– кожна координата розбивається на базову частину (пропонується розглядати бази довготи і широти окремо: B_X і B_Y), яку записують тільки один раз для безперервного відносно великого набору координат за допомогою спеціального тега типу BASE, і другу частину, яка являє собою допустимі молодші біти кожної координати, що не міститься в базі;

– виключення координат, що повторюються більше одного разу поспіль, можна шляхом введення спеціальної мітки REP (скорочено R), яка вказує на кількість повторень попередньої координати (така ситуація досягається, коли об'єкт стоїть на одному місці);

– подання строкових подань (які від початку є координатами точки та її складовими частинами) у числовому вигляді, тобто перехід від символьного кодування до двійкового;

– використання традиційних програмних забезпечень для архівації перед остаточним розміщенням на сервері архівації файлів.

Всі ці кроки забезпечують ефективне стиснення кореневої інформації перед зберіганням у централізованому файловому архіві. Наступним кроком у цій роботі є реалізація остаточного рішення як повноцінного програмного продукту.

3 ДОСЛІДЖЕННЯ СИСТЕМИ

3.1 Обґрунтування вибору засобів розробки

У попередньому розділі було розроблено алгоритм для ефективної мінімізації маршрутних даних, і цей алгоритм буде надалі реалізовано у програмному додатку.

Перед початком реалізації програми необхідно визначити низку концептуальних питань, пов'язаних з програмою;

– який метод програмування найкраще підходить для реалізації цього алгоритму в заданих умовах;

– яку мову програмування, що підтримують обрану технологію програмування, найбільше підходить для реалізації програмного забезпечення в поточній ситуації;

– яке середовище розробки (або простий набір окремих інструментів розробки) найкраще підходить для даної програмної реалізації.

3.1.1 Вибір технології розробки

Тому основною проблемою, що стоїть практично перед усіма розробниками програмного забезпечення, є вибір моделей і технологій для його розробки. Найбільш поширеними у виробництві на сьогоднішній день є технології структурного (процедурного) програмування та об'єктно-орієнтованого програмування. Ці дві технології мають свої специфічні переваги та недоліки.

Процедурне програмування, відоме також як структурне, ґрунтується насамперед на використанні окремих структурних блоків, які називаються підпрограмами (процедурами і функціями).

Історично перші комп'ютерні програми були відносно простими і мали пакетний режим роботи. Вони приймали певну інформацію на вході (можливо, на перфокартах), виконували певні операції для обробки даних і видавали результат. При цьому вихідні дані були функціонально тісно пов'язані з вхідними – рис. 3.1.

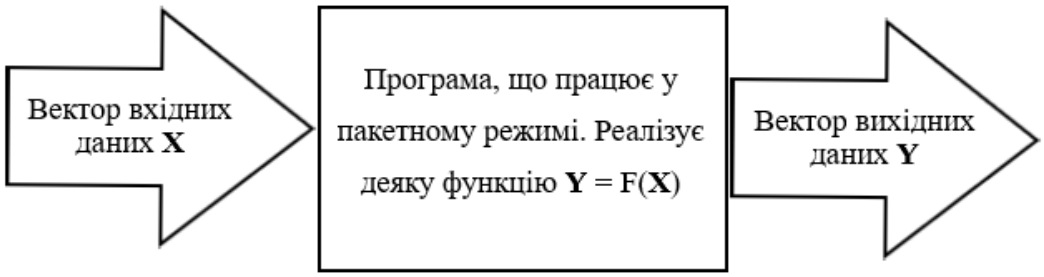


Рисунок 3.1- Схема роботи пакетної програми

Незважаючи на відсутність чіткого зв'язку між функцією і структурою, більшість пакетних програм мали простий лінійний порядок виконання. Це означає, що підпрограм було небагато, або, принаймні, використання підпрограм не було важливим елементом методології програмування.

З ускладненням функціональності програмного забезпечення змінювалася і його внутрішня структура. Поступово склалася схема взаємодії між користувачем і програмою рис. 3.2. Програми почали запитувати інформацію та активно реагувати на поведінку людини. Збільшення функціональності відповідно збільшило складність програмного коду. Пересічний програміст тепер може швидко і легко розібратися в незнайомому коді без особливих хитрощів. Розміщення коду в простому лінійному порядку без виділення великих блоків стало недоречним, головним чином з метою розуміння цього коду.

Тут варто звернути увагу на психологічні особливості сприйняття людиною складних "великих" завдань. Неструктуровані "великі" завдання (наприклад, написання дипломної роботи) часто викликають своєрідну психологічну млявість, що призводить до поступової нездатності впоратися із завданням. Людям зручно розбивати проблему на меншу кількість (зазвичай близько десятка, бажано три-чотири) завдань (наприклад, написання розділу дисертації) і не думати про виконання кожного з них. Після того, як проблема чітко зрозуміла на найвищому рівні абстракції, слід почати розробляти під задачі. Кожне під завдання слід розбити на окремі "під задачі", тобто менші під задачі на нижчому рівні. Після такої

декомпозиції всі ці під задачі слід проаналізувати. На певному етапі наступні під задачі можна не де компонувати, а реалізувати безпосередньо в програмному коді.

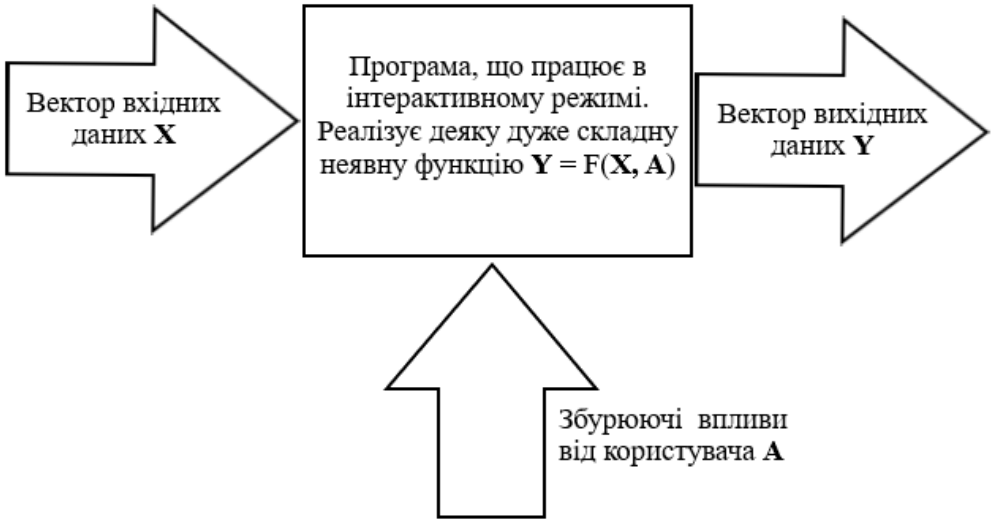


Рисунок 3.2 - Схема для додатків, які активно взаємодіють з користувачем.

Іншими словами, розвиток технології програмування йшов паралельно з розвитком користувацьких інтерфейсів. Загалом, інтерфейси командного рядка відповідають парадигмі структурного програмування.

Більш формально, структуроване програмування означає створення програм за трьома основними принципами: слідування, розгалуження та повторення.

Послідовність означає виконання операторів або блоків програмного коду один за одним. Розгалуження реалізується за допомогою різних умовних операторів, таких як if, які дозволяють вибрати один з декількох варіантів виконання програми. Повторення зазвичай спричинене зацикленням (повторенням тієї самої ділянки коду кілька разів поспіль), але той самий принцип можна застосувати і до підпрограм.

Загалом, структурне програмування іноді називають застарілою технікою програмування, яка була витіснена об'єктно-орієнтованим програмуванням разом з віконними інтерфейсами (деякі сучасні мови програмування загального призначення навіть не дозволяють створювати структуровані програми, а лише

об'єктно-орієнтовані - наприклад, Visual C# та Java). Однак використання цієї техніки програмування абсолютно виправдане при створенні невеликих програм (наприклад, до 10 000 рядків коду і без будь-яких передбачуваних розширень), а відповідний код набагато краще розпізнається, ніж об'єктно-орієнтована версія.

Суть об'єктно-орієнтованого підходу до програмування полягає в тому, що система розглядається як сукупність об'єктів, окремих сутностей, які мають власний набір властивостей зі своїми внутрішніми параметрами і можуть взаємодіяти за допомогою викликів методів, які є деякими діями (або, дещо більш опосередковано, шляхом надсилання повідомлень, які обробляються методами об'єкта; для цього потрібен активний об'єкт для доставки повідомлення одержувачу, наприклад, віконний менеджер Windows.).

У програмному коді для роботи з об'єктом необхідно спочатку створити об'єкт. Об'єкт створюється як змінна, типом якої є клас об'єкта. Клас описує, якими властивостями володіє об'єкт даного типу (тобто яку інформацію він може зберігати) і якими методами він володіє (тобто які дії він може виконувати). Об'єкт - це набір значень, що описують, чому дорівнюють властивості цього об'єкта (кожен об'єкт отримує метод від свого класу, тобто всі об'єкти, які належать цьому класу, мають один і той самий метод).

Навіщо потрібен такий спеціальний підхід, якщо самі об'єкти не слугують жодній меті (навпаки, введення об'єктів ускладнює програму і вводить нові сутності)? Тому що, згідно з алгоритмом, програма може спростити своє розуміння, реалізувавши всі необхідні для її роботи сутності у вигляді класів і об'єктів; саме тому ОО-підхід рекомендується для великих проектів (десятки тисяч рядків коду і більше), оскільки він дає змогу розуміти програму більш ефективно. Розбиваючи програму на об'єкти і проектуючи її класи, можна сказати, що розуміння предметної області (у разі "великих" проектів) наближається до нормального людського мислення. Люди мислять у термінах класів, об'єктів і відносин між ними.

На рисунку 3.3 порівнюється складність проектів з однаковою функціональністю, але побудованих різними способами (структурний та об'єктно-орієнтований підходи), залежно від їхнього розміру. З графіка на рисунку 3.3 видно,

що якщо необхідно реалізувати продукт з невеликою функціональністю (тобто де кількість рядків коду, що реалізують її, явно невелика, близько декількох тисяч рядків), то видно, що класи краще не використовувати. Якщо ж додаток має більш-менш важливу функціональність і реалізується хоча б у кількох тисячах рядків коду, то має сенс розглянути можливість використання об'єктно-орієнтованого підходу. Програма, що містить понад 10 тис. рядків коду, обов'язково має бути побудована на принципах ООП.

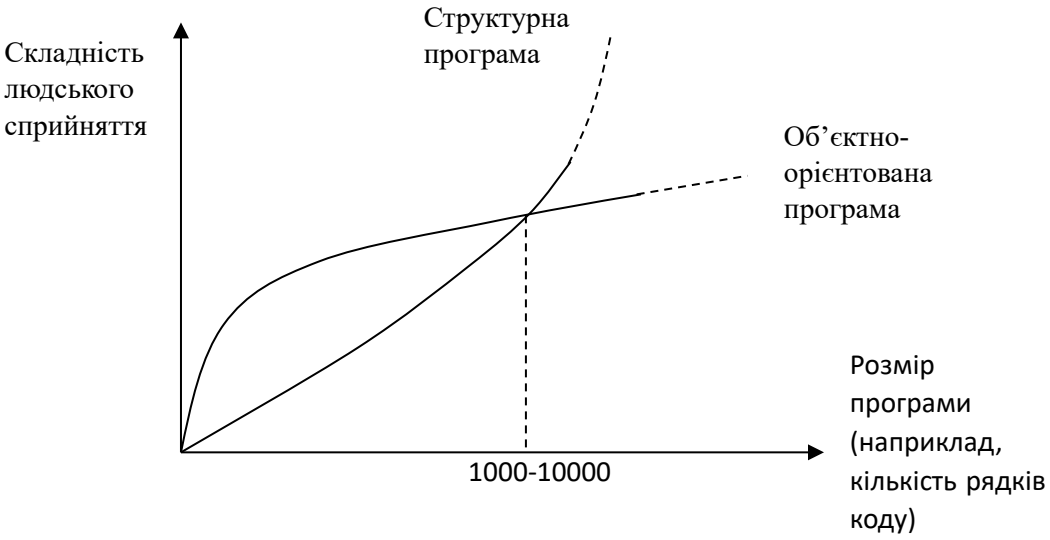


Рисунок 3.3 – Порівняння складності коду двох програм

Крім наведених вище міркувань, слід зазначити, що на вибір підходу до програмування часто впливають й інші чинники, як-от: можливість майбутніх удосконалень, створення найзрозумілішого коду (для всієї команди, що працює над проектом, а не лише для одного програміста) або просто бажання замовника застосувати найсучасніший підхід до програмування (або, навпаки, такий, що допускає використання старих, застарілих прийомів).

ОО-підхід вимагає поділу (декомпозиції) всієї предметної області на об'єкти (класи) і визначення зв'язків між об'єктами, а також дотримання трьох основних принципів: інкапсуляції, успадкування і поліморфізму.

Інкапсуляція - це поєднання даних (значень властивостей класу в даному об'єкті) і засобів, що використовуються для їх обробки (методів класу). Це також є психологічно доцільним: цілісні сутності, що називаються класами, які обробляють власні дані, можуть бути реалізовані окремо. Доступ до об'єктів цих класів здійснюється за допомогою методів, які складають інтерфейси класів.

Успадкування дуже корисне, тому що дає змогу значно скоротити кількість повторюваного коду (до чого завжди потрібно прагнути під час розроблення будь-якого програмного забезпечення). Згідно з цим принципом, клас із загальним набором властивостей і методів виокремлюється в кілька досконаліших класів. Цей клас оголошується батьківським і є базовим для кількох похідних класів (дочірніх, успадкованих). Кожен клас-нащадок успадковує всі свої властивості та методи від базового класу, але перед цим у нього залишаються власні властивості та методи.

Наприклад, клас Student є похідним від класу Person, де кожен Person має властивості FirstName і LastName та метод Rest(). Однак клас Student має свої властивості та методи, відмінні від Person: AverageGrade, Number of Medicine, PassExam() тощо.

Під час успадкування призначення методу в батьківському та похідному класах може бути однаковим, а реалізація - різною. Такі методи називаються перевантаженням. Наприклад, метод Rest() у класі Person реалізовано для відпочинку на дивані, а в класі Student - для походу в клуб. Слід ще раз підкреслити, що призначення методу в обох випадках однакове.

Поліморфізм - це здатність функції приймати об'єкти як з базового, так і з похідних класів, а також викликати перевантажені методи класу, переданого у функцію. Це досить специфічна особливість, і багато програмістів часто використовують підхід ОО, не вдаючись до поліморфізму.

Наприклад, припустимо, що у вашій програмі є функція SpendWeekend(). Нехай аргументом цієї функції є об'єкт класу Person. Тоді в цю функцію можна передавати об'єкти всіх класів, похідних від Person: Student, Employee, Pensioner тощо, оскільки всі вони є Humans (спадкоємцями цього класу). Зрозуміло, що ця функція має містити різні дії: CleanApartment(), GoToMarket() і Rest(). Тож за

рахунок поліморфізму, просто вказавши всередині цієї функції ім'я методу Rest(), не вказуючи, який клас викликати, вже під час виконання програми, якщо у функцію передається об'єкт класу Student, буде викликано його метод Rest(), якщо у функцію передається об'єкт класу Pensioner, то автоматично викликається його метод Rest() і т. д. Про функцію SpendWeekend() кажуть, що вона поліморфна, оскільки реалізує принцип поліморфізму.

Іншими важливими поняттями ООП є статична приналежність класів, абстрактні методи і класи, дружба між функціями і класами.

Виходячи з характеристик двох перерахованих вище наявних методів програмування, структурний підхід обрано як простіший метод, що задовольняє як малий розмір проектованого програмного забезпечення (непромислове), так і вимогу складності (програма не є складною, тому що операції, що виконуються, є однотипними: здебільшого це рядкові операції).

3.1.2 Вибір мов програмування

Після вибору технології програмування наступним кроком є вибір мови програмування (або сімейства мов, наприклад, у разі написання програмного продукту у вигляді розподіленого клієнт-серверного додатку).

У найзагальнішому розумінні всі мови програмування можна розділити на засоби розробки десктопних додатків, засоби веб-розробки та засоби мобільної розробки, залежно від типу платформи, на якій буде створено кінцевий програмний продукт. Мобільні технології взагалі не застосовні до цього завдання, оскільки програмний продукт, що розробляється, не вимагає мобільності, а обчислювальні ресурси мобільної системи набагато менші, ніж у настільної або, тим більше, клієнт-серверної системи. Що стосується вибору між "веб-додатками" і "десктопним програмним забезпеченням", то перший варіант видається більш прийнятним з наступних міркувань:

– є повністю кросплатформеним (високоякісні веб-додатки однаково добре працюють у будь-якому сучасному веб-браузері, незалежно від операційної системи);

– можна використовувати обчислювальні ресурси серверів, які набагато потужніші за настільні ПК;

– може мати досить швидкий і зручний інтерфейс, якщо в ньому використовуються сучасні веб-технології.

Таким чином виберемо веб-розробку і проаналізуємо наявні засоби які можуть бути використані.

Спектр сучасних засобів розроблення Інтернет-ресурсів настільки великий, що проаналізувати їх усі в рамках цієї роботи абсолютно неможливо. Тому в процесі вибору враховуватимуться як об'єктивні фактори (наприклад, підтримка новітніх технологій програмування), так і суб'єктивні, що мають широкий спектр впливу (наприклад, індивідуальні схильності до тієї чи іншої мови або стилю програмування і, відповідно, популярність відповідного засобу розробки на масовому рівні) буде похідною від обох цих факторів.

Таким чином, розглянемо найпоширеніші засоби розробки, які зараз популярні в індустрії.

Весь процес веб-розробки традиційно ділиться на два елементи: front-end і back-end, приклад рисунок 3.4.

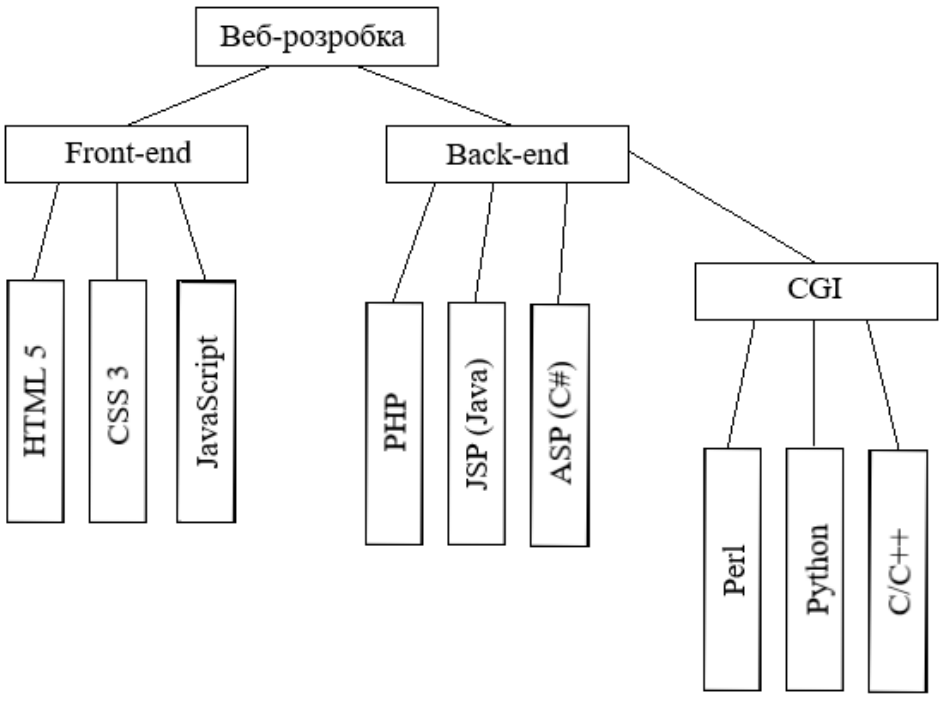


Рисунок 3.4 - Засоби веб-розробки

3.1.2.1 Засоби розробки Front-end

Простіше кажучи, front-end у програмуванні - це те, що бачить користувач. Для настільних систем це інтерфейс, а back-end - логіка роботи програми. Теоретично (як це часто і робиться, особливо в Linux-системах) він може бути реалізований окремою програмою з текстовим консольним інтерфейсом. Завдання front-end - зібрати всі вхідні дані, необхідні для того, щоб консольна утиліта back-end виконувала осмислені операції в зручному для широкого кола користувачів вигляді. У багатьох випадках front-end і back-end поєднані в одному програмному забезпеченні. Це характерно для настільних систем.

Якщо говорити конкретніше про веб-програмування, то тут ситуація максимально біполярна (навіть більше, ніж описана вище ситуація, коли один програміст розробляє інтерфейс для внутрішньої утиліти, створеної іншим розробником). Це пов'язано з тим, що користувач запускає і "бачить" систему через браузер на віддаленій машині (так званий клієнт), тоді як логіка програми і майже вся необхідна інформація знаходиться на абсолютно окремому комп'ютері-сервері.

Таким чином, весь програмний код і текст, який виконується в браузері, тобто на стороні клієнта, називається front-end частиною. З іншого боку, все, що виконується на сервері, називається back-end складовою.

Не вдаючись у детальний аналіз, можна сказати, що набір інструментів для front-end розробки значно менший, ніж для back-end розробки. До них належать:

– HTML - версія 5 (HTML5) - це найсучасніший стандарт мови розмітки гіпертексту, де під гіпертекстом традиційно розуміють текст із гіперпосиланнями на інші частини документа, які можуть супроводжуватися зображеннями, аудіо, відео тощо;

– CSS - версія 3 (CSS3) - правила визначення стилів для різних елементів веб-сторінки, точніше каскадні таблиці стилів;

– JavaScript - це найпотужніший інструмент front-end розробки інтерфейсів, який виводить html-сторінки на новий рівень майже повноцінних додатків, що активно реагують на дії користувача та надсилають і отримують інформацію від нього.

Замінити ці технології неможливо, тому розглянемо їх докладніше.

– Hyper Text Markup Language (HTML) - мова розмітки гіпертексту - використовується для опису всіх документів World Wide Web (WWW).

HTML-документ - це текстовий файл зі спеціальними позначками (тегами), які передаються з комп'ютера-сервера в браузер клієнта за запитом клієнта і використовуються браузером для відображення вмісту файлу на екрані комп'ютера.

За допомогою цих міток можна виділяти заголовки документа, змінювати колір, розмір і написання тексту, вставляти графічні зображення і таблиці. Однак головною перевагою гіпертексту перед звичайним текстом є можливість додавання гіперпосилань у вміст документа. Гіперпосилання - це спеціальна структура мови HTML, що дає змогу за клацанням миші відобразити інший документ.

Сама мова HTML є варіантом більш загальної мови XML (eXtensible Markup Language - розширювана мова розмітки), яка, з одного боку, звужує її можливості, а з іншого - розвиває їх. Так, якщо XML дозволяє використовувати довільні теги, тобто теги з довільними назвами, то в HTML назви тегів досить фіксовані і не допускають використання слів, відмінних від стандартних. З іншого боку, вміст веб-сторінки, розміщений поруч з певним тегом (точніше, між парами однакових тегів), матиме властивості, визначені стандартом (на відміну від тексту, розміщеного поруч з тегом XML, який сам по собі майже не має сенсу і використовується тільки за наявності певного тегу) і буде відображатися браузером відповідно до значення цих тегів: якщо тег<B>, то текст відображається напівжирним шрифтом, якщо<i>, то курсивом.

HTML-документ складається з двох компонентів: власне текстову інформацію - дані, що складають зміст документа - і теги, які є спеціальними конструкціями мови HTML, що використовуються для розмітки документа і керування його представленням. Теги HTML визначають спосіб відображення тексту, який компонент виступає в ролі гіпертекстового посилання, а також які графічні та мультимедійні об'єкти включаються в документ.

Графічна та звукова інформація, що міститься в HTML-документах, зберігається в окремих файлах. Програма, яка використовується для перегляду

HTML-документів, називається браузером, що інтерпретує теги розмітки та маніпулює текстом і графікою, розміщуючи їх на екрані в потрібному порядку. Файли, що містять HTML-документи, мають розширення . htm або .html.

У більшості випадків теги використовуються парами. Пара складається з початкового тегу<ім'я_тега>та кінцевого тегу</ім'я_тега>. Дія парного тегу починається, коли зустрічається початковий тег, і закінчується, коли зустрічається відповідний кінцевий тег. У більшості випадків пара, що складається з початкового і кінцевого тегів, називається контейнером, а частина тексту між початковим і кінцевим тегами називається елементом.

Послідовність символів, з яких складається текст, може складатися з пробілів, табуляції, переходу на новий рядок, повернення каретки, кирилиці, латиниці, розділових знаків, цифр і спеціальних символів (#, +, $, @ і т.д.), за винятком наступних чотирьох символів, які мають особливе значення в HTML: < (менше), >(більше), & (амперсанд), "(лапки)", лістинг 3.1. Якщо ці спеціальні символи повинні бути включені в текст, вони повинні бути закодовані спеціальною послідовністю символів:

Лістинг 3.1 - Винятки символів

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

Першим з тегів HTML є однойменний тег <html>, який завжди повинен стояти в останньому рядку документа. Ці теги вказують на те, що рядки між ними є єдиним гіпертекстовим документом. Без цих тегів браузери та інші програми перегляду не можуть визначити формат документа і правильно його інтерпретувати.

У більш загальних рисах, HTML-документ складається з двох частин, заголовка (head) і тіло (body), лістинг 3.2, розташованих у наступному порядку:

Лістинг 3.2 – Приклад розташованих заголовка та тіла

```
<Html>
```

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

У більшості випадків назва документа містить пару тегів <title> ... </title>. Багато програм перегляду використовують їх як заголовок вікна, в якому відображається документ. Програми, які індексують документи в Інтернеті, використовують заголовок для ідентифікації сторінки. Хороший заголовок має бути достатньо довгим, щоб правильно вказувати на відповідну сторінку, а також бути включеним у заголовок вікна. Назви документів вставляються між тегами відкриття і закриття title.

Тіло документа є найважливішим елементом, оскільки в ньому міститься весь вміст веб-сторінки. Тіло розміщується між тегами<body>….</body>. Усе, що розміщується між цими тегами, інтерпретується браузером відповідно до правил мови HTML для коректного відображення сторінки на екрані монітора.

HTML-текст ділиться на абзаци за допомогою тега ; тег поміщається на початок кожного абзацу, і коли програма перегляду знаходить цей тег, вона розділяє абзаци порожнім рядком. Використання закриваючого тега необов'язкове.

Якщо необхідно "перервати рядок", перенісши решту тексту на новий рядок без створення нового абзацу, використовується тег переведення рядка
. Цей тег виводить наступний за ним текст на новий рядок. На відміну від тега абзацу, тег
не розриває рядки. Для цього тега не існує парного закривального тега.

Мова HTML підтримує як логічне, так і фізичне форматування вмісту документа. Логічне форматування визначає призначення певного тексту, тоді як фізичне форматування визначає зовнішній вигляд тексту.

При логічному форматуванні тексту браузер виділяє різні частини тексту відповідно до структури документа. Для відображення заголовка використовуйте один з тегів заголовка. Заголовки в типовому документі поділяються на рівні: Мова HTML дозволяє визначити шість рівнів заголовків: h1 (заголовки першого рівня),

h2, h3, h4, h5 і h6. Заголовки першого рівня зазвичай більші і щільніші, ніж заголовки другого рівня приклад можна побачити на лістингу 3.3.

Лістинг 3.3 - Приклади використовуваних тегів заголовків

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

Теги фізичного форматування безпосередньо визначають спосіб відображення тексту на екрані браузера. наприклад, пари напівжирний текст,<u> </u>підкреслений текст і керують шрифтом тексту. Ці елементи вважаються застарілими, і специфікація HTML5 рекомендує використовувати замість них стилі, описані нижче.

Тег <img> вставляє зображення в документ так, як якщо б це був один великий символ, лістинг 3.4. Зображенням не потрібен закриваючий тег.

Лістинг 3.4 - Приклад використання тегів

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

Для створення гіпертекстового посилання використовується пари тегів <a> ... для створення гіпертекстового посилання. Фрагменти тексту, зображення та інші об'єкти, розміщені між цими тегами, відображаються у вікні браузера як гіпертекстові посилання. При активізації таких об'єктів шляхом натискання на них мишею у вікні браузера завантажується новий документ або відображається інша частина поточної веб-сторінки. Гіпертекстові посилання створюються за наступною формулою, лістинг 3.5.

Лістинг 3.5 – Створення гіпертекстових посилань

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

Href - обов'язковий атрибут, значенням якого є URL-адреса запитуваного ресурсу; при вказівці значення атрибута href лапки не потрібні (як і при вказівці

значення будь-якого іншого атрибута в будь-якому тезі, але настійно рекомендується використовувати подвійні або одинарні лапки). При вказівці посилання на документ на іншому сервері гіперпосилання повинне мати такий вигляд як у лістингу 3.6.

Лістинг 3.6. - Посилання на документ

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

Узагальнюючи функціональність мови HTML, можна сказати, що різні теги можна використовувати для малювання таблиць, форматування тексту та вставки зображень, відео- та аудіофайлів у документи. Протягом свого розвитку теги <style> та пов'язані з ними каскадні таблиці стилів (Cascading Style Sheets) ставали дедалі більшим центром уваги.

Каскадні таблиці стилів (CSS) - це мова, яка містить набір властивостей для опису зовнішнього вигляду документа; специфікація CSS (наразі CSS3) визначає мову опису для зв'язування властивостей і характеристик з елементами документа. Розуміння таблиць стилів має вирішальне значення для додавання динамічних стилів до сторінки. Динамічна стилізація в цьому контексті означає використання скриптів для внесення змін до таблиць стилів, прив'язаних до документа.

На вузлі консорціуму W3C (www.w3.org) можна ознайомитися з останньою інформацією про нововведення та елементи, підтримувані таблицями стилів.

Таблиця стилів - це абстракція, що визначає стиль документа незалежно від його змісту та структури. Веб-майстрам доступні три способи додавання таблиць стилів у документи - як правило, з підвищенням рівня складності збільшуються і пропоновані можливості, оскільки одночасно зростає ступінь абстракції. Перший спосіб полягає у використанні вбудованих таблиць стилів. Вбудовані стилі задаються безпосередньо на елементі. Другий спосіб - визначення стилів на початку документа за допомогою глобальних таблиць стилів. Третій, найбільш абстрактний і потужний спосіб - це визначення стилів окремо в іншому документі за допомогою пов'язаних таблиць стилів.

Внутрішні стилі дещо відрізняються від традиційного HTML. При використанні внутрішніх стилів важко змінити зовнішній вигляд документа. Перевага такого підходу полягає в тому, що потрібно менше розмітки, а HTML можна використовувати більш повно для представлення вмісту презентації. За допомогою глобальних таблиць стилів можна ефективніше розділити презентацію і вміст, а також швидко і незалежно змінювати стилі документа і додатків. Пов'язані таблиці стилів надають додаткові переваги при представленні контенту у вигляді серії сторінок або визначенні всього веб-сайту в одному файлі.

Термін "каскадний" у назві CSS означає можливість об'єднання різних таблиць стилів для створення єдиного визначення стилю для елемента або всього документа. Це дозволяє передбачувано поєднувати таблиці стилів веб-сайту з таблицями стилів документа і навіть внутрішніми стилями.

Таким чином, вбудований стиль - це, по суті, таблиця стилів, визначена безпосередньо в тегу елемента для одного екземпляра елемента. Вбудовані таблиці стилів визначаються за допомогою атрибута STYLE, а дані атрибута визначаються за допомогою мови таблиці стилів. Нижче у лістингу 3.7, наведено HTML-код, який збільшує шрифт і вирівнює абзац по центру на жовтому тлі для відображення вмісту абзацу:

Лістинг 3.7- Приклад збільшення шрифту та вирівняння абзац по центру

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

Внутрішні стилі корисні, коли ви вивчаєте CSS і вам потрібно швидко змінити один екземпляр елемента. Однак внутрішні стилі не відповідають ідеології структурованого документа і погано працюють, коли потрібно змінити зовнішній вигляд великої кількості елементів у документі, де представлення і зміст повністю відокремлені. Щоб відокремити стиль документа від його структури, таблиці стилів

повинні бути визначені в заголовку документа або у вигляді окремого файлу, пов'язаного з документом.

Наступний, більш ідеологічно правильний варіант - використовувати окремий тег<STYLE>для позначення глобальної таблиці стилів, який зазвичай розміщують у заголовку документа. Розміщення всіх стилів документа в одному місці полегшує зміну режиму відображення документа. Наведений нижче приклад (лістинг 3.8) таблиці стилів визначає вигляд усіх абзаців у документі. Щоб змінити режим відображення абзацу, просто змініть STYLE. За допомогою внутрішніх стилів кожен абзац у документі потрібно змінювати окремо.

Лістинг 3.8 – Вигляд усіх абзаців у документі

```
<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>
```

Використовується селектор, щоб пов'язати стиль із певним елементом. У наведеному вище прикладі створено простий селектор, який посилається на стиль у кожному абзаці. Можна також визначити більш функціональні контекстні селектори, про які йтиметься далі в цьому розділі.

Ще одним із трьох способів визначення стилів є використання пов'язаних таблиць стилів у зовнішніх файлах. Перевага використання пов'язаних таблиць стилів полягає в тому, що всі правила та стилі можуть бути визначені та розміщені в одному файлі, який може бути використаний на багатьох сторінках і сайтах. Пов'язані таблиці стилів дають змогу в одному документі змінити оформлення всіх абзаців на сайті. Пов'язані таблиці стилів кешуються локально на диску клієнта окремо від документа, тому кожен документ стає меншим, а продуктивність підвищується, оскільки інформацію про стилі потрібно завантажувати один раз.

Щоб визначити пов'язану таблицю стилів, розміщується тег <LINK> у шапку документа, лістинг 3.9.

Лістинг 3.9 - Розміщення тега <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 ідентифікує пов'язаний файл як таблицю стилів, атрибут TYPE визначає MIME-тип таблиці стилів, а атрибут HREF є URL-адресою, що вказує на зовнішню таблицю стилів. Пов'язані таблиці стилів можуть містити тільки контекстні правила та визначення стилів і не можуть містити HTML-код.

Під час створення таблиць стилів у документі використовується той самий синтаксис, що й під час створення пов'язаних таблиць стилів: мова CSS складається із селекторів і правил подання. Селектори визначають елементи, які мають бути пов'язані з певними правилами, а правила подання визначають, як ці елементи мають оброблятися.

У CSS існує два типи селекторів: прості та контекстні. Прості селектори об'єднують елементи на основі їхніх атрибутів або типу, незалежно від їхнього контекстного положення всередині інших елементів Контекстні елементи є потужнішими і дають змогу асоціювати правила з контейнерами певних елементів, усі теги<EM>всередині тега<P>.

У базовому вигляді простий селектор може бути створений для того, щоб пов'язати певний елемент, клас елемента або ідентифікатор (ID) з певним стилем. Наведений у лістингу 3.10, код ілюструє деякі прості селектори та правила їх відображення.

Лістинг 3.10 - Прості селектори

```
<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 розташовані через кому, що дає змогу описувати кілька селекторів одночасно як показано у лістингу 3.11.

Лістинг 3.11 – Селектори

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

Контекстні селектори визначають ієрархію контейнерів, з якими асоціюється стиль. Ієрархія контейнерів визначається порядком проходження елементів у списку, розділених комами. Наприклад, таке твердження визначає правило для всіх елементів EM в елементі P, лістинг 3.12.

Лістинг 3.12 – Визначення правила для всіх елементів EM

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

Кожен селектор може посилатися на тег CLASS, ID або тип елемента. Нижче у лістингу 3.13 наведено складніший варіант контекстного селектора.

Лістинг 3.13 - Складний варіант контекстного 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.

Псевдокласи складаються з елементів одного типу, що задовольняють певним контекстним критеріям. Наприклад, елемент Anchor є псевдокласом visited. Активні та невідвідувані посилання є псевдокласами active і link відповідно. Псевдокласи розділяються в таблицях стилів двокрапкою: приклад у лістингу 3.14.

Лістинг 3.14 – Псевдокласи в таблицях стилів

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

У другому прикладі ім'я елемента (A) опущено, оскільки тільки посилання є псевдокласом посилань. Псевдокласи можна використовувати так само, як класи та ID-покажчики, і вони не залежать від регістру.

Кілька селекторів можуть належати до одного й того самого елемента; у CSS визначено каскадний порядок, який використовується для обробки перехресних селекторів і областей дії правил. Каскадний порядок об'єднує всі правила, що застосовуються до елемента, відсортовані відповідно до їхнього визначення. Наприклад, елемент Strong всередині елемента H1 може мати правила відображення, визначені селектором H1, селектором STRONG і контекстним селектором елемента Strong всередині елемента H1. Каскадні параметри CSS визначають порядок об'єднання цих трьох правил. У загальному випадку порядок об'єднання цих трьох правил визначається каскадними параметрами CSS. У загальному випадку правила конкретнішого контекстного селектора скасовують правила менш конкретного селектора, причому пріоритет мають правила, підпорядковані у вихідній таблиці стилів або документі.

Детально розглянувши каскадні таблиці стилів, слід зупинитися і на основних особливостях мови програмування JavaScript - сучасного і дуже потужного засобу розробки зовнішніх інтерфейсів. Це спеціалізована мова програмування, яку традиційно використовують для керування об'єктною моделлю документа в браузері під час перегляду інтернет-сторінок (існують продукти, що

перетворюють JavaScript на мову загального призначення, команди якої виконуються на сервері, наприклад Node.JS). Хоча такий підхід сильно суперечить початковій концепції використання цієї мови програмування, значна кількість програмістів розглядає можливість перетворення JavaScript і на серверну мову, враховуючи наявність низки традиційних засобів розробки back-end.

Особливістю JavaScript є те, що його попередній програмний код є інтерпретованим.

Оператори в цій мові, загалом схожій на мову С, розділяються крапками з комою. Мова чутлива до регістру, що часто не береться до уваги початківцями, імовірно, тому, що HTML, який часто використовують у поєднанні з JavaScript, не чутливий до регістру (імена тегів і атрибутів HTML можуть бути як малими, так і великими літерами).

Однорядкові коментарі виділяються символом //, а багаторядкові - парою символів /* і */.

До даних застосовується слабка (динамічна) перевірка типу. Для операторів з даними різних типів, останні автоматично перетворюються до потрібного типу. Типи даних бувають примітивні та складені. Примітивні типи містять прості однорідні значення. Такі дані можна передавати в якості параметрів функцій як значення, а не посилання. Складені типи містять різнорідні дані (в тому числі складені), які можна передавати у функції тільки як посилання.

JavaScript - об'єктно-орієнтована мова, але заснована не на класах, а на прототипах. Існує чотири типи об'єктів: вбудовані об'єкти, об'єкти браузера, об'єкти документа та об'єкти користувача (програміста).

Введення-виведення практично обмежується тільки документом і взаємодією з користувачем. За замовчуванням вважається, що доступ до локальної файлової системи заборонено. Однак браузери можуть надавати спеціальні об'єкти, які можуть маніпулювати файловою системою користувача, хоча і з попередженнями про небезпеку виконання файлових операцій.

Сценарії JavaScript активно взаємодіють з об'єктами, вбудованими у веб-сторінки. У цьому й полягає призначення скриптів. Однак перш ніж ця взаємодія

відбудеться, код скрипта має бути впроваджений у текст HTML-документа - існує кілька способів зв'язати HTML-документ із конкретним скриптом, зазвичай просто помістивши його в тег-контейнер<SCRIPT>, тобто між дескрипторами<script>i</script>.

Контейнер<SCRIPT>у цьому випадку розміщується безпосередньо в будь-якому місці HTML-документа. Програмний код може бути записаний безпосередньо в HTML-документі або в спеціальному текстовому файлі, який можна викликати з основного HTML-документа. Почнемо з першого варіанта. Спочатку браузер знаходить у тілі веб-документа тег<script>і намагається обробити весь наступний за ним текст як код сценарію. Потім він продовжує роботу доти, доки не зустріне закриваючий тег</script>. Усі наступні символи вважаються HTML-текстом. Будь-який HTML-документ може містити будь-яку кількість "скриптових включень", але кожний скрипт повинен починатися і закінчуватися відповідним тегом. Функціональність усієї веб-сторінки може залежати від її положення в тілі HTML-документа, про що йтиметься нижче.

Теги-контейнери <script>можуть містити атрибут SRC, який вказує ім'я або URL текстового файлу, що містить код скрипта. Цей атрибут необхідний, якщо скрипт міститься в окремому файлі, а не безпосередньо в HTML-документі. Розширення файлу скрипта може бути будь-яким, але зазвичай використовується js як показано у лістингу 3.15.

Лістинг 3.15 - Розширення файлу

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

Якщо скрипт міститься в окремому файлі, то теги<SCRIPT>i</ SCRIPT>, природно, не записуються. Сценарії, що завантажуються із зовнішніх файлів, можна вважати просто вставленими в HTML-документ.

У браузерах, які можуть підтримувати скриптинг, користувач може відключити цю функціональність (особливо з метою безпеки). Але деякі браузери взагалі не підтримують роботу зі сценаріями. У таких ситуаціях бажано хоча б

виводити повідомлення про те, що на сторінці присутній сценарій, але в даному конкретному випадку він не виконується. Для цього в HTML-документах використовуються теги-контейнери<noscript>. Контейнерний тег містить текст, який виводиться користувачеві, якщо з якоїсь причини скрипт не виконується. Усі браузери, що підтримують роботу зі сценаріями, ігноруватимуть вміст тега<noscript>, якщо тільки робота зі сценаріями не була відключена. І навпаки, браузери, які не підтримують скриптинг, ігнорують вміст тега <script> (особливо якщо він вкладений у тег <!-- -->, який є HTML-коментарем) і в відобразять тег <noscript>.

Приклад наведено у лістингу 3.16.

Лістинг 3.16 – Відображення тега у браузері.

```
<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 повідомлення виводиться в невеликому діалоговому вікні за допомогою функції оповіщення.

Як уже зазначалося, бібліотеки функцій (визначення функцій) і скрипти, що використовуються в декількох HTML-документах на одному або декількох сайтах, зазвичай розміщуються в окремих файлах. Скрипти також можуть бути записані у вигляді рядків тверджень, розділених крапками з комою. Такі рядки укладаються в лапки і слугують, наприклад, значенням атрибута події, лістинг 3.17.

Лістинг 3.17 – Атрибут подій.

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

Виклик функції та її визначення можуть слідувати в будь-якому порядку, але тільки якщо вони знаходяться в одному контейнері <script>. Якщо вони знаходяться в різних контейнерах <script>, то визначення функції має передувати виклику. Аналогічно, якщо визначення функції міститься в іншому файлі, то воно має бути завантажене в HTML-документ до виклику функції.

Якщо завантажити некоректну версію HTML-коду і спробувати її запустити, то з'явиться діалогове вікно з написом «Error: object expected». Браузер інтерпретує HTML-теги по порядку. Наприклад, зустрічаючи вираз виклику функції myfunc() в одному контейнері<script>, браузер ще не має в пам'яті визначення цього об'єкта (функції) в іншому контейнері<script>. Якщо визначення функції та її виклик знаходяться в одному контейнері<script>, то спочатку в пам'ять завантажується її вміст, а потім проводиться аналіз. Коли інтерпретатор зустрічає виклик функції, у пам'яті шукається її визначення і, в разі успіху, виконується код функції.

Якщо ви хочете, щоб браузер відображав скрипт до завантаження HTML-елементів документа, то краще помістити скрипт на початок HTML-коду або в контейнер<head>заголовка документа.

3.1.2.2 Засоби Back-end розробки

Як уже зазначалося вище, back-end - це програмні компоненти, що працюють на сервері, і вибір інструментів для реалізації цих компонентів дуже широкий рис. 3.4, праворуч.

Усі засоби серверного веб-програмування можна розділити на два класи: автономні програми, що використовують технологію CGI, і мови, у яких вихідний код вбудовується безпосередньо у веб-сторінку і виконується через препроцесор веб-сервера перед відображенням у браузері клієнта. Другий підхід зручніший, що видно зі зменшення кількості CGI-інструментів. На зміну їм приходять повноцінні продукти, такі як:

– Java Server Pages - JSP активно просувається шанувальниками цієї відкритої та сучасної (generic) мови програмування;

– активні серверна сторінка Microsoft ASP (звідси надзвичайно високий рівень підтримки цього рішення);

– використання мови PHP є кращим для багатьох розробників серверних додатків, що зумовлено різними перевагами цього продукту.

Наразі PHP є найпоширенішою мовою веб-програмування. Переважна більшість веб-сайтів і веб-сервісів в Інтернеті написані з використанням PHP. За деякими оцінками, PHP використовується у понад 80% веб-сайтів, включаючи такі сервіси, як facebook.com та baidu.com. Така популярність не дивна - PHP є простою мовою, яка дозволяє швидко і легко створювати веб-сайти і портали різної складності.

PHP був створений датським програмістом Расмусом Лердорфом 1994 року і спочатку являв собою набір скриптів на Perl, іншій мові програмування PHP іншій мові програмування. Пізніше цей набір скриптів було переписано в інтерпретатор мови C. З моменту своєї появи PHP перетворився на корисний набір інструментів для зручного створення веб-сайтів і веб-додатків.

Розглянемо переваги PHP:

– найпоширеніші операційні системи (Windows, MacOS і Linux) мають свої версії пакетів розробки PHP;

– PHP працює з різними веб-серверами: наприклад, Apache, Nginx, IIS;

– ця мова вирізняється простотою і легкістю вивчення. Як правило, за невеликого досвіду програмування на PHP можна створювати прості сайти;

– PHP схожий на C, тому якщо ви знаєте C або мову з C-подібним синтаксисом, вивчення PHP не складе труднощів;

– PHP підтримує інтеграцію з багатьма системами баз даних (наприклад, MySQL, MS SQL Server, Oracle, Postgres, MongoDB);

– послуги хостингу поширені і дешеві. Зазвичай хостингові компанії розміщують PHP-сайти на веб-серверах Apache або Nginx, що працюють на операційних системах Linux. Оскільки і веб-сервер, і операційна система на базі Linux є безкоштовними, загальна вартість хостингу знижується;

– PHP постійно розвивається, тому з виходом нових версій додаються нові функції, які дозволяють мові програмування справлятися з новими завданнями. І, як правило, перехід на нову версію ніколи не викликає труднощів;

– існує безліч якісних інтернет-ресурсів, що підтримують програмування на PHP, де можна поставити свої запитання й отримати правильні відповіді в найкоротші терміни.

З огляду на всі переваги, основним засобом розробки back-end у цьому проєкті буде PHP.

Таким чином, мовами програмування для обраного напряму (веб-розробка) обраємо PHP і JavaScript.

Останнім питанням є вибір конкретного середовища розробки для написання коду на обраних мовах. Такими середовищами можуть бути популярна серед сучасної студентської спільноти Visual Studio Code, більш спеціалізована PHP Storm або простий текстовий редактор, наприклад, Sublime Editor. Поширена перевага таким інструментам пояснюється простотою поставленого завдання (зокрема, відсутністю необхідності використання систем управління базами даних тощо) та стандартністю (поширеністю) обраної мови програмування серед представників сучасної спільноти програмістів.

3.2 Особливості реалізації елементів обраного алгоритму, що мають відношення до мінімізації маршрутних даних

Розглянемо особливості операції 1-8 з рис. 2.7.

Операція 1, завантажує файл, що містить нестиснуту маршрутну інформацію, і не має жодних спеціальних функцій, крім відкриття файлу в текстовому режимі.

Операції 2 усі числа мають бути зчитані з файлу і записані заново з меншою кількістю знаків після коми (п'ять замість шести). Для форматованого виведення у файл можна використовувати, наприклад, функцію fprintf.

Операції 3, пов'язана з переміщенням сільськогосподарської техніки по полю можна пропустити, вибравши за замовчуванням пункт 4 в таблиці 2.2.

Операція 4 виконується таким чином:

– якщо під час зчитування чергової координати X її базовий біт не збігається з поточним X- базою, то у вихідний файл записується новий тег B_X;

– якщо під час читання чергової Y-координати її базовий біт не збігається з поточною Y-базою, то у вихідний файл записується новий тег B_Y;

– у будь-якому разі біт про базу відкидається у всіх координатах.

Операція 5 виконується відповідно до алгоритму, блок-схема якого наведена на рис. 3.5

При зчитуванні чергової координати X вона порівнюється з попереднім значенням X', і якщо воно збігається, то координата Y порівнюється з попереднім значенням. Якщо обидві координати збігаються, то ці координати не повинні записуватися у вихідний файл, а лічильник збігів повинен бути збільшений на одиницю. Якщо після скидання значення лічильника попереднього порівняння не дорівнює нулю, то у вихідний файл має бути додано відповідну кількість повторень тега REP.

Під час виконання операції 6 достатньо зчитати координати з вхідного файлу у вигляді рядка і записати їх у вихідний файл у вигляді чисел відповідного формату (1-байтові довгі цілі числа). Ці операції можна виконати за допомогою функції fprintf, як і в пункті 2.

Операція 7 може бути виконано з використанням стандартних методів архівування, таких як алгоритм ZIP.

Операція 8 полягає у збереженні отриманого файлу в потрібному місці на сервері.

Таким чином, видно, що тільки на кроці 4 і (більшою мірою) на кроці 5 присутні елементи реального алгоритму. Загалом у цьому підрозділі описано особливості реалізації всіх восьми елементів алгоритму мінімізації маршрутних даних, розробленого в підрозділі 2.4.

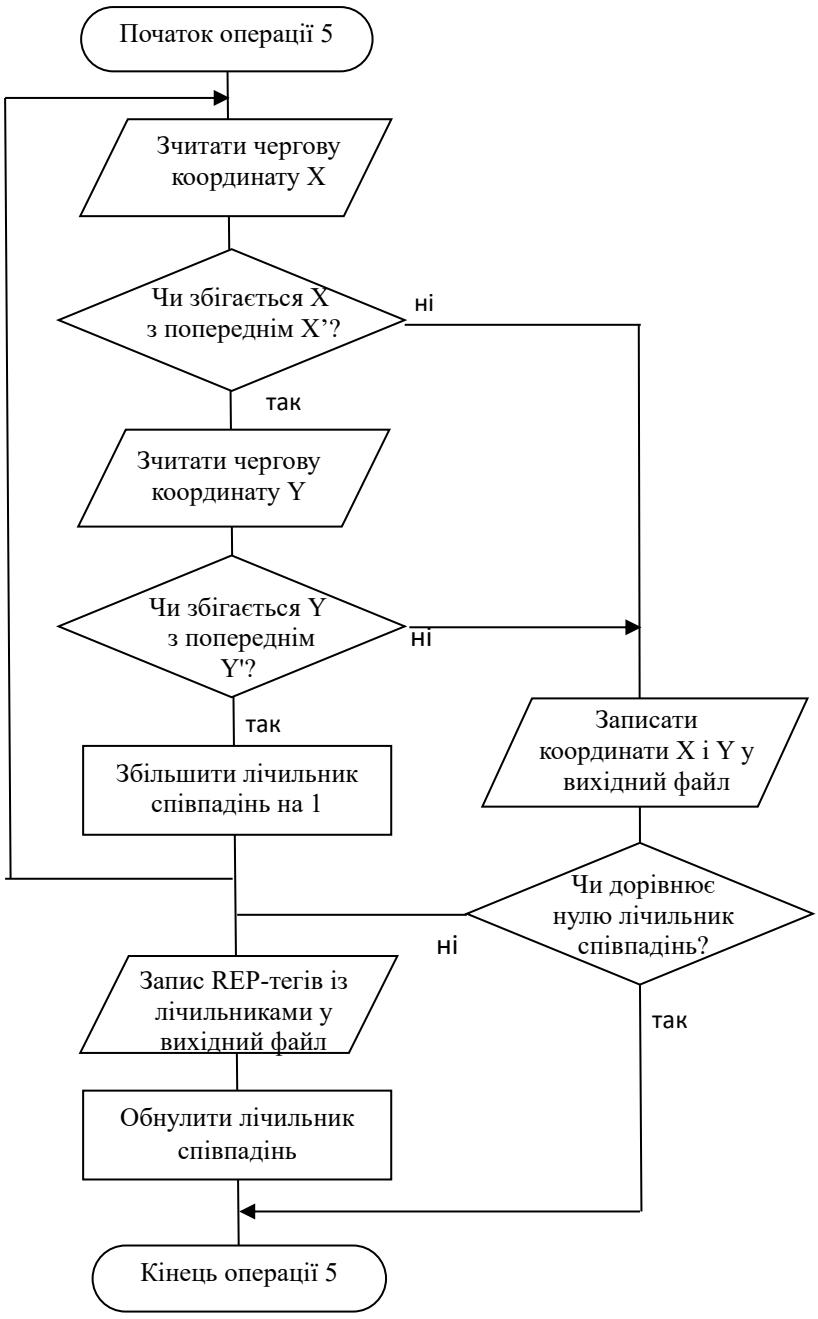


Рисунок 3.5 - Блок-схема загального алгоритму стиснення маршрутних даних

3.3 Визначення формату зберігання маршрутної інформації

Особливостям зберігання маршрутної інформації вже було приділено багато уваги, тому можна зробити висновок, що розроблено унікальний формат зберігання відповідної інформації. Розглянемо це більш детально.

Спочатку, на початку підсумкового файлу в окремому рядку записується дата і час початку запису маршрутної інформації. Потім в окремому рядку вказується інтервал, з яким отримують дані про положення відстежуваного мобільного пристрою (стандарт - 1 секунда).

Потім на окремому рядку напишіть тег типу BASE:

– B_X – вказання бази для першої координати X;

– B_Y – вказання бази для другої координати Y.

Праворуч від цих міток вказується, наприклад, числове значення розрядів, що містяться у відповідній базі даних:

– B_X 50.382;

– B_Y 38.405.

Потім пари координат X і Y описуються в окремих рядках, причому в числовому вигляді кодуються тільки їхні молодші значущі розряди (які не входять до бази).

За наявності повторюваних координат файл містить мітку REP (або її скорочення R), за якою слідує кількість повторень. Тут N однакових рядків відповідають N-1 повторенням.

Таким чином, можна домогтися високої ефективності підсистеми стиснення, здійснюючи процес мінімізації маршрутної інформації описаним вище способом і зберігаючи дані в описаному форматі, що в ідеалі повинно бути досліджено експериментально на основі експлуатованого програмного продукту і має відповідати його опису.

3.4 Особливості реалізації в конкретних технічних рішеннях розробленої системи мінімізації маршрутних даних

Щоб забезпечити покрокову інтеграцію, весь код був написаний під керівництвом універсального програмного продукту Denwer, що включає в себе

веб-сервер (Apache), СУБД (MySQL або MareaDB), інтерпретатор PHP, мови Perl і поштового сервера, а також портативна версія, яка може бути завантажена безпосередньо з флеш-пам'яті.

Усі розроблені вище алгоритми було реалізовано мовою PHP з використанням JavaScript, а результати представлено на рис. 3.6.

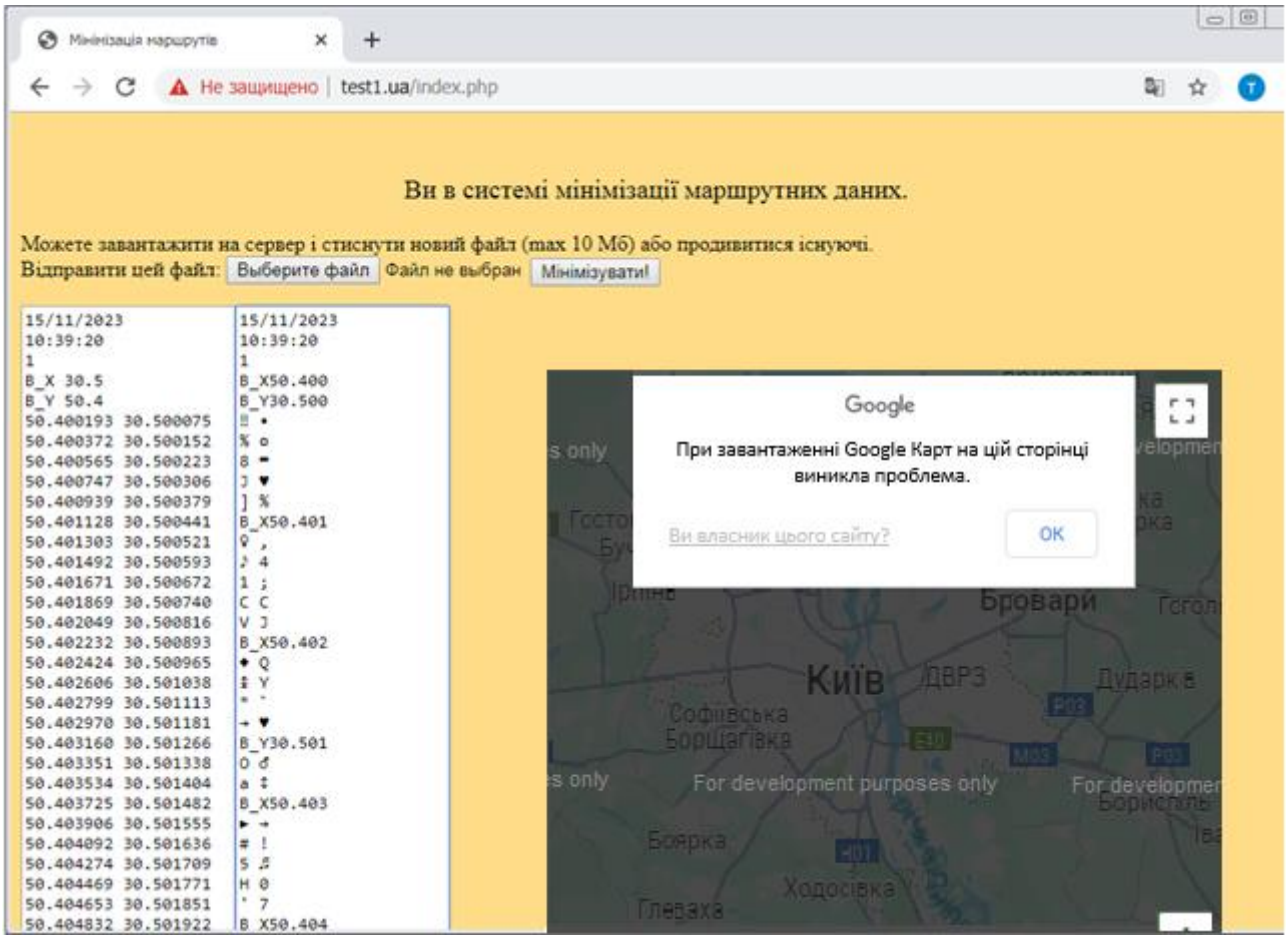


Рисунок 3.6 - Стиснення файлу маршрутної інформації розробленою системою

Ви побачите, що файли можуть бути завантажені в систему за допомогою стандартної кнопки "Вибрати файл". Обраний файл відправляється з локального комп'ютера на сервер, де відбувається його стиснення, результат якого відображається у вертикальній текстовій області праворуч і, найголовніше, передається до архівної папки для постійного зберігання.

У правій частині вікна програми видно спробу відображення маршрутів на Google Maps, але з середини 2018 року ця послуга стала платною. Безкоштовний доступ пропонується за 200 доларів США на місяць, але реєстрація все одно вимагає введення реальних даних кредитної картки, і найнеприємніше, що якщо на зареєстрований акаунт якимось чином надходить більше 200 доларів США на місяць (а це 20 000 запитів на місяць), то різниця списується з рахунку абонента шляхом автоматичного списування з рахунку абонента. Таким чином, у разі DDoS-атаки сайт, що використовує API Google Maps, наприкінці місяця може автоматично опинитися винен Google тисячі доларів (або більше). Таким чином, використання цієї системи для некомерційних ресурсів на даному етапі небезпечно тільки з фінансового погляду.

Загалом система мінімізації маршрутної інформації реалізована і готова до практичного використання.

3.5 Аналіз продуктивності системи

Однак ефективність цього процесу може бути різною, здебільшого залежно від обраного варіанта подання бази даних (п. 2 у табл. 2.2 чи п. 4, розглянутий у попередньому розділі). На рис. 3.6 показано приклад використання бази даних із трьома десятковими знаками, у результаті якого файл маршрутної інформації з початковим розміром 2149 КБ було стиснуто до 901 КБ, тобто приблизно до 42% від первісного об'єму, а після застосування архівації зменшився майже вдвічі - до 487 КБ. Загальний коефіцієнт стиснення становить приблизно 22% від початкового розміру, що майже на 10% краще, ніж при використанні простого архівування.

Отже, розроблена в цій роботі система показує достатній рівень економічної ефективності (тобто без великих фінансових вкладень) і дає змогу економити щонайменше 10% дискової ємності.

3.6 Висновки до розділу 3

В цьому розділі описується розробка та реалізація системи мінімізації маршрутної інформації.

Користувачі можуть з будь-якої точки світу підключатися до розробленої системи через Інтернет і завантажувати файл з маршрутною інформацією, який автоматично стискається, результати відображаються користувачеві, а сам стиснутий файл зберігається в архівній папці, що з організаційного погляду є зручністю. Розроблена система економить щонайменше на 10% більше дискового простору, ніж просте архівування, і тому рекомендується для використання в практичній діяльності.

ВИСНОВКИ

У магістерський роботі розроблено систему мінімізації маршрутної інформації на основі нового алгоритму. Для того, щоб використати можливість скорочення маршрутної інформації, в роботі проаналізовано існуючі методи збору, перетворення та зберігання маршрутної інформації. Розроблено комплексний алгоритм мінімізації маршрутної інформації на основі відомих алгоритмів та запропоновано необхідні супутні науково-технічні рішення (моделювання маршрутної інформації, створення унікального формату зберігання даних тощо).

Також було проведено програмну реалізацію розробленого алгоритму та дослідження ефективності, яке показало збільшення ступеня стиснення не менше ніж на 10% порівняно з традиційними методами. У дослідженні враховано різні залежності (зокрема, швидкість роботи об'єкта), а також вимоги надійності зберігання та захисту відповідної маршрутної інформації.

Розроблений програмний продукт працездатний та може бути використаний в реальних додатках.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке backend-розробка і чим вона відрізняється від frontend. URL: https://www.buh24.com.ua/shho-take-backend-rozrobka-i-chym-vona-vidriznyayetsya-vid-frontend/.

2. Anderson R. J. Security engineering: a guide to building dependable distributed systems. Wiley & Sons, Incorporated, John, 2010. 1088 p.

3. Fox C. Data science for transport: a self-study guide with computer exercises. Springer, 2019. 204 p.

4. Front-end: що це таке та чим відрізняється від Backend розробки? | Wezom. IT-компания полного цикла разработки программных продуктов WEZOM - Киев, Украина. URL: https://wezom.com.ua/ua/blog/chto-takoe-front-end-razrabotka.

5. Hall R. Handbook of transportation science. Springer London, Limited, 2006.

6. Introduction to algorithms / T. H. Cormen et al. MIT Press, 2009. 1320 p.

7. Ovaska S. J., Laplante P. A. Real-Time systems design analysis: tools for the practitioner. Wiley-IEEE Press, 2017. 600 p.

8. Toth P., Vigo D. Vehicle routing: problems, methods, and applications. Philadelphia : Society for Industrial and Applied Mathematics, 2014. 463 p.

9. Оптимізація вантажних потоків. StudFiles. URL: https://studfile.net/preview/5607420/.

10. Ashcroft S., Hillier F. S., Lieberman G. J. Introduction to operations research. Operational research quarterly (1970-1977). 1975. Vol. 26, no. 4. P. 881. URL: https://doi.org/10.2307/3009248 (date of access: 23.11.2023).

11. Ord K., Taha H. A. Operations research: an introduction. Operational research quarterly (1970-1977). 1972. Vol. 23, no. 2. P. 234. URL: https://doi.org/10.2307/3008276 (date of access: 23.11.2023).

12. Salomon D. Data compression: the complete reference. 4th ed. Springer, 2006. 1092 p.

13. Shi Y. Q. Image and video compression for multimedia engineering: fundamentals, algorithms, and standards. 2nd ed. Boca Raton : CRC Press, 2008. 541 p.

14. Talbi E.-G. Metaheuristics: from design to implementation. Wiley & Sons, Incorporated, John, 2009.

15. Toth P., Vigo D. Vehicle routing: problems, methods, and applications. Philadelphia : Society for Industrial and Applied Mathematics, 2014. 463 p.

16. Traveling salesman problem: a computational study / R. E. Bixby et al. Princeton University Press, 2011. 606 p.