



**Міністерство освіти і науки України
Одеський національний морський університет
Кафедра кібербезпеки та захисту інформації**

КОНСПЕКТ ЛЕКЦІЙ
з дисципліни «Програмна обробка та аналіз даних»
другого (магістерський) рівня вищої освіти
за спеціальністю F5 Кібербезпека та захист інформації
галузі знань F Інформаційні технології
Розділ 1. Базові структури та формати даних

Одеса, 2025



Міністерство освіти і науки України
Одеський національний морський університет
Кафедра кібербезпеки та захисту інформації

КОНСПЕКТ ЛЕКЦІЙ
з дисципліни «Програмна обробка та аналіз даних»
другого (магістерський) рівня вищої освіти
за спеціальністю F5 Кібербезпека та захист інформації
галузі знань F Інформаційні технології
Розділ 1. Базові структури та формати даних

Затверджено
на засіданні кафедри КБЗІ
Протокол № 2 від 06.10.2025 р.

Одеса, 2025

Конспект лекцій з дисципліни «**Програмна обробка та аналіз даних**» Розділ 1. «Базові структури та формати даних» для другого (магістерський) рівня вищої освіти за спеціальністю F5 Кібербезпека та захист інформації галузі знань F Інформаційні технології / Укл.: *Лебедєва О.Ю.* – Одеса, 2025. – 38с.

Укладач: **Лебедєва О.Ю.**, к.т.н., доц.

ЗМІСТ

Змістовий модуль 1. Базові структури та формати даних	5
Лекція 1. Введення у аналіз даних	5
1. Типи аналізу даних	5
2. Процес аналізу даних	7
3. Основні методи аналізу даних	8
Лекція 2. Типи та формати даних	11
1. Типи даних	11
2. Формати даних	11
3. Формати даних XLS та CSV	13
4. Формати даних XML та JSON	15
Лекція 3. Структури даних	22
1. Основні визначення	22
2. Масиви і робота з ними	22
3. Зв'язний список	25
Лекція 4. Структури даних (Продовження)	31
1. Стеки	31
2. Черги	33
Література	38

Змістовий модуль 1. Базові структури та формати даних

Лекція 1. Введення у аналіз даних

1. Типи аналізу даних
2. Процес аналізу даних
3. Основні методи аналізу даних

1. Типи аналізу даних

Аналіз даних – це процес збору, очищення й моделювання даних для отримання корисної інформації. На основі цих даних потім формуються звіти, які допомагають приймати стратегічні рішення.

Саме аналіз даних відіграє вирішальну роль, оскільки допомагає особам, які приймають рішення, зрозуміти й осмислити зібрану інформацію. Уявіть, що у вас є безліч фрагментів головоломки (дані) і ви хочете побачити всю картину повністю (аналітичні висновки). Аналіз даних схожий на складання фрагментів цієї головоломки в єдине ціле (тобто перетворення даних на знання), щоб виявити те, що є важливим.

Простий приклад аналізу даних: щоразу, коли ми приймаємо будь-яке рішення в нашому повсякденному житті, ми думаємо про те, що сталося минулого разу або що станеться, вибравши це конкретне рішення. Це не що інше, як аналіз нашого минулого чи майбутнього та прийняття рішень на його основі. Для цього ми збираємо спогади про наше минуле або мріємо про наше майбутнє. Тож це не що інше, як аналіз даних. Тепер те саме, що робить аналітик для бізнес-цілей, називається *аналізом даних*.

Існує багато різних типів аналізу даних. Кожен із них застосовується для пошуку відповідей на окремі запитання.

- Описовий аналіз даних;
- Діагностичний аналіз;
- Прескриптивний аналіз;
- Предиктивний аналіз;
- Когнітивний аналіз;
- Аналіз тексту.

Описовий аналіз даних – це процес вивчення минулих даних для виявлення тенденцій, закономірностей та ключових показників, який відповідає на запитання «Що сталося?» за допомогою методів агрегації, систематизації та візуалізації даних. Його основна мета – отримати кількісний опис досліджуваного масиву даних, використовуючи статистичні показники, такі як середнє, медіана, мода, дисперсія та стандартне відхилення.

Описовий аналіз (також відомий як описова статистика) зосереджується на описі та агрегуванні даних, що були зібрані. Це основа для розуміння минулих показників, що дозволяє отримати загальне уявлення про досліджуваний набір даних.

Приклади застосування:

- Щомісячні звіти про ефективність продажів;
- Аналіз результатів опитувань задоволеності клієнтів;
- Оцінка показників веб-трафіку та залучення користувачів;
- Підрахунок, скільки DDoS-атак було на сервер упродовж тижня;
- Підрахунок кількості спроб входу за добу тощо.

Відмінність від інших типів аналітики: описовий аналіз відповідає на запитання «Що сталося?». Він є першим кроком перед діагностичним аналізом («Чому це сталося?»), передбачуваним аналізом («Що може статися?») та наказним аналізом («Що робити далі?»), які представляють вищі рівні аналітики.

Діагностичний аналіз даних – це процес дослідження даних для виявлення причин певних подій, закономірностей та зв'язків, який доповнює описовий аналіз. Його мета — зрозуміти "чому" сталося те, що спостерігається в даних, на відміну від простого опису "що"

сталось. Цей тип аналізу допомагає виявляти причини подій, структурувати дані та встановлювати зв'язки між різними показниками.

Зазвичай діагностичний аналіз передбачає виявлення аномалій у даних (наприклад, несподіваних змін показників), збір даних, пов'язаних із цими аномаліями, і застосування статистичних методів для пошуку можливих пояснень.

Приклади використання:

- Компанія може використати діагностичний аналіз, щоб з'ясувати, чому знизилися продажі певного продукту;
- Кореляція підозрілих логів: аномальні IP-адреси + неуспішні входи;
- Виявлення джерела зараження — наприклад, шкідливий файл, завантажений через фішинговий лист тощо.

Предиктивний аналіз (прогностичний аналіз). На відміну від попередніх типів аналізу, які досліджують отримані дані та роблять висновки про минуле, передиктивна аналітика використовує дані для прогнозування майбутнього. Відповідає на запитання «що може статися в майбутньому?». Для аналізу передбачень спочатку історичні дані обробляються, а потім на основі отриманих результатів робляться припущення, що відбудеться в майбутньому. Наприклад, так можна виявити певні тенденції й оцінити ймовірність їх повторення.

Наприклад, допомагає помітити, що мобільний додаток для схуднення активніше завантажують навесні, а взимку – кількість завантажень падає. Отже, міграцію з однієї бази даних до іншої краще запланувати на зиму — від можливих збоїв постраждає менше користувачів.

Ще один приклад: модель прогнозує, що ймовірність фішингової кампанії проти компанії зросте наступного тижня, оскільки за останні 3 дні різко збільшилась кількість підозрілих листів із одного домену.

Прескриптивний аналіз допомагає відповісти на запитання, що потрібно зробити, щоб досягнути певної мети. Організації використовують результати такого аналізу, щоб за умов невизначеності приймати рішення, ґрунтуючись на даних. Фахівці, що виконують аналіз, часто покладаються на машинне навчання, щоб знайти закономірності у великих семантичних моделях і оцінити ймовірність різних результатів.

Наприклад, після аналізу атак система рекомендує перевести сервери на інший сегмент мережі та включити додаткове шифрування.

Когнітивний аналіз – це складна форма аналізу даних, яка виходить за рамки традиційних методів. Під час такого аналізу застосовуються технології машинного навчання та обробки природної мови. Це дає змогу розуміти інформацію, вибудовувати логічні зв'язки та навчатися в спосіб, подібний до людського мислення.

Ціль когнітивного аналізу – імітувати людський спосіб мислення, щоб отримувати глибше розуміння, виявляти закономірності й складати передбачення.

Аналіз тексту – це спосіб навчити комп'ютери розуміти людську мову. Він охоплює використання алгоритмів та інших методів для отримання інформації з великого обсягу текстових даних, наприклад дописів у соцмережах чи відгуків клієнтів.

Аналіз тексту допомагає фахівцям збирати інформацію про те, що говорять люди, і знаходити закономірності. Отримані дані дають змогу приймати виважені рішення у сферах бізнесу, маркетингу та досліджень.

Наприклад:

Тип аналізу	Питання	Приклад у кібербезпеці
Описовий	Що сталося?	Підрахунок кількості атак
Діагностичний	Чому це сталося?	Визначення джерела атаки
Дослідницький	Що ще можна знайти?	Виявлення аномального користувача
Прогностичний	Що станеться далі?	Прогноз фішингової кампанії
Прескриптивний	Що робити далі?	Автоматичне блокування IP

2. Процес аналізу даних

Збір та інтерпретація даних для прийняття рішень – це деталізований процес, що потребує системного підходу.

Процес аналізу даних це не що інше, як збір інформації за допомогою відповідної програми чи інструменту, який дозволяє вам досліджувати дані та знаходити в них закономірності. На основі цієї інформації та даних ви можете приймати рішення або робити остаточні висновки.

Процес аналізу даних складається з наступних етапів:

- Збір вимог до даних;
- Збір даних;
- Очищення даних;
- Аналіз даних;
- Інтерпретація даних;
- Візуалізація даних.

Збір вимог до даних – перш за все, ви повинні подумати, чому ви хочете зробити цей аналіз даних? Усе, що вам потрібно, щоб з'ясувати мету чи мету проведення аналізу даних. Ви повинні вирішити, який тип аналізу даних ви хочете зробити! На цьому етапі ви повинні вирішити, що аналізувати і як це вимірювати, ви повинні зрозуміти, чому ви досліджуєте і які заходи ви повинні використовувати для цього аналізу.

Збір даних – після збору вимог ви отримаєте чітке уявлення про те, що вам потрібно виміряти та що мають бути ваші висновки. Тепер настав час зібрати ваші дані відповідно до вимог. Зібравши дані, пам'ятайте, що зібрані дані потрібно обробити або впорядкувати для аналізу. Оскільки ви збираєте дані з різних джерел, ви повинні вести журнал із зазначенням дати збору та джерела даних.

Очищення даних – тепер будь-які зібрані дані можуть бути некорисними або невідповідними для вашої мети аналізу, тому їх слід очистити. Дані, які збираються, можуть містити повторювані записи, пробіли або помилки. Дані мають бути очищені та без помилок. Цю фазу потрібно виконати перед аналізом, оскільки на основі очищення даних ваш результат аналізу буде ближчим до очікуваного результату.

Аналіз даних – коли дані зібрані, очищені та оброблені, вони готові до аналізу. Коли ви маніпулюєте даними, ви можете виявити, що маєте точну інформацію, яка вам потрібна, або вам може знадобитися зібрати більше даних. Під час цієї фази ви можете використовувати інструменти аналізу даних і програмне забезпечення, яке допоможе вам зрозуміти, інтерпретувати та зробити висновки на основі вимог.



Рисунок 1.1 – Інструменти аналізу даних

Інтерпретація даних – після аналізу даних нарешті настав час інтерпретувати результати. Ви можете вибрати спосіб вираження або передачі аналізу даних або просто словами, або, можливо, таблицею чи діаграмою. Потім скористайтеся результатами процесу аналізу даних, щоб визначити найкращий спосіб дій.

Візуалізація даних дуже поширена у повсякденному житті; вони часто з'являються у формі діаграм і графіків. Іншими словами, дані представлені графічно, щоб людському мозку було легше їх зрозуміти й обробити. Візуалізація даних часто використовується для виявлення невідомих фактів і тенденцій. Спостерігаючи за зв'язками та порівнюючи набори даних, ви можете знайти спосіб отримати значущу інформацію.

Розглянемо приклад процесу аналізу даних.

Формулювання задачі: виявити підозрілу активність у корпоративній мережі, щоб запобігти витоку даних. Питання: чи можна за логами входів користувачів визначити потенційні спроби зламу?

Збір даних: логи систем аутентифікації, наприклад, в форматі CSV/JSON з полями: timestamp, user, ip, status (success/fail), device.

Очищення та підготовка даних: видаляємо дублікати записів, заповнюємо відсутні дані (наприклад, відсутній user → "unknown"), перетворюємо часові мітки у стандартний формат, створюємо додаткові атрибути: hour (година входу) та fail_count (кількість невдалих входів поспіль).

Аналіз даних: наприклад, будемо графіки: кількість успішних/невдалих входів по годинах; топ-10 IP з найбільшою кількістю спроб; виявляємо аномалії (наприклад, користувач заходить уночі, хоча зазвичай тільки вдень); визначаємо записи, які відхиляються від стандарту (наприклад, багато невдалих входів за короткий час).

Інтерпретація результатів: наприклад, аналізуємо які IP позначені як підозрілі, які користувачі відхиляються від звичайної поведінки.

Впровадження: наприклад блокувати IP при >10 невдалих спробах за 5 хв; повідомляти адміністратора.

3. Основні методи аналізу даних

Є два основні способи обробки даних: кількісний і якісний аналізи.

Кількісний аналіз передбачає роботу з числами та вимірюваннями (наприклад, аналіз результатів опитування, отриманих за допомогою оцінок). Під час кількісного аналізу застосовуються лише математичні й статистичні методи. Цей аналіз відповідає на запитання типу "скільки".

Кількісний аналіз спирається на структуровані методи збору даних, такі як опитування, експерименти та спостереження для збору числових даних. Потім ці дані систематизуються, аналізуються та інтерпретуються з використанням статистичних інструментів для отримання значних висновків. Цей метод дозволяє дослідникам перевіряти гіпотези, робити прогнози та отримувати корисну інформацію, засновану на

емпіричних даних.

У кількісному аналізі використовуються різні статистичні методи, включаючи описову статистику, логічну статистику, регресійний аналіз та перевірку гіпотез. Описова статистика узагальнює та описує характеристики набору даних, такі як середнє значення, медіана та стандартне відхилення. З іншого боку, статистичні дані, що ґрунтуються на висновках, дозволяють дослідникам робити узагальнення про популяцію на основі вибірки. Регресійний аналіз допомагає виявити взаємозв'язки між змінними, тоді як перевірка гіпотез підтверджує чи спростовує дослідницькі гіпотези.

Якісний аналіз передбачає розуміння суб'єктивного значення нечислових даних. Наприклад, він підійде для аналізу відповідей під час співбесіди чи розпізнавання емоцій на фотографіях. Під час якісного аналізу виявляються закономірності, теми та інша корисна інформація, і його головна мета – отримати точну та детальну інформацію.

На відміну від кількісного аналізу, що спирається на чисельні виміри та статистичні методи, якісний аналіз вивчає контекст, нюанси та складність даних для отримання багатих, контекстуально обґрунтованих висновків.

Якісний аналіз ґрунтується на переконанні, що дані не можуть бути зведені до простих чисел. Натомість вона прагне зрозуміти глибинні смисли, мотивації та культурні контексти, які формують людську поведінку та досвід. Цей підхід особливо добре підходить для дослідження складних соціальних, культурних та психологічних явищ, які нелегко піддаються кількісній оцінці.

Дослідники використовують різні методи збору даних для якісного аналізу, включаючи інтерв'ю, фокус-групи, спостереження учасників та контент-аналіз тексту, зображень та аудіозаписів. Ці методи дозволяють дослідникам збирати докладні розповіді, описи та погляди учасників, що дозволяє глибше вивчити їхній досвід.

Якісний аналіз ідеально підходить для дослідницьких проєктів з невеликими розмірами вибірки. Це дозволяє проводити поглиблений аналіз обмеженої кількості випадків, забезпечуючи детальне розуміння точки зору кожної окремої людини.

Методи вимірювання якісних даних: юзабіліті-тестування, теплові карти, фокус-групи, дослідження щоденників, запис сесій та відгуки користувачів.

Наприклад, організації потрібні чітко визначені категорії впливу, загрози та вразливості. Тоді можна ввести три категорії безпеки на основі потенційного впливу на організацію, якщо відбудуться певні події, які поставлять під загрозу ІТ-активи, необхідні організації для виконання призначеної місії, захисту її активів, виконувати свої юридичні обов'язки, підтримувати свої повсякденні функції та захищати осіб.

Категорія 1. Низька. Очікується, що вона матиме обмежений негативний вплив на діяльність організації, організаційні активи або окремих осіб.

Категорія 2. Помірна або середня. Очікується, що вона матиме серйозний негативний вплив на діяльність організації, організаційні активи або окремих осіб.

Категорія 3. Висока. Очікується, що вона матиме серйозний або катастрофічний негативний вплив на діяльність організації, активи організації або окремих осіб.

Наприклад, правоохоронна організація, яка керує надзвичайно чутливою слідчою інформацією, визначає, що потенційний вплив втрати конфіденційності є високим, потенційний вплив втрати цілісності є помірним, а потенційний вплив втрати доступності є помірний.

SC = {(конфіденційність, ВИСОКА), (цілісність, ПОМІРНА),
(доступність, ПОМІРНА)}.

Запитання для самоперевірки

1. Що таке аналіз даних?

2. Які типи аналізу даних ви знаєте?
3. З яких етапів складається процес аналізу даних?
4. Які інструменти використовуються при аналізі даних?
5. Які є способи обробки даних?
6. Які є методи кількісного аналізу?
7. Які є методи вимірювання якісних даних?

Лекція 2. Типи та формати даних

1. Типи даних
2. Формати даних
3. Формати даних XLS та CSV
4. Формати даних XML та JSON

1. Типи даних

Типи даних визначають характер інформації (наприклад, текст, числа, графіка), її властивості та операції, що можна з нею виконувати.

Типи даних класифікують за їхнім змістом та призначенням:

- Текстові – рядки символів, як-от заголовки, параграфи, звіти та документи;
- Числові – числа, з якими можна виконувати математичні операції;
- Графічні – дані у вигляді зображень;
- Аудіо- та відео – звукові та відеозаписи;
- Структуровані – дані, організовані в таблиці, бази даних або ієрархії (наприклад, у форматі JSON чи XML).
- Геопросторові – дані про розташування географічних об'єктів.

Текстові дані складаються із заголовків, параграфів, реквізитів, також можуть включати таблиці та ілюстрації. Прикладами текстових даних є нормативно-правові акти, розпорядчі документи, положення, програми, звіти, угоди. Для роботи з ними найчастіше використовуються редактори Microsoft Word, LibreOffice Writer, Google Документи. Рекомендовані формати оприлюднення текстових даних: DOC, DOCX, ODT, TXT, RTF та HTML.

Графічні дані – якщо набір даних є фотографією чи зображенням, будьте певні – ви маєте справу з графічними даними. Прикладом таких даних можуть бути фото архівних документів, генеральні плани міст тощо. Графічні дані зазвичай публікуються у відкритих форматах PNG, JPG чи JPEG. Часто буває, що текстові чи навіть структуровані дані оприлюднюються у вигляді графічних. Тоді для розпізнавання тексту потрібно використовувати технології OCR (Оптичне розпізнавання тексту).

Структуровані дані включають таблиці та ієрархічні дані, які містяться в базах даних. Це найбільш цінний тип відкритих даних, оскільки їх легко автоматично обробляти.

Геопросторові дані описують положення географічних об'єктів на території. Наприклад, містобудівна документація, супутникові знімки, дані інтерактивних мап тощо. Існує два типи геопросторових даних: векторні (SHP, GeoJSON, KML, MID, MIF, XML) та растрові (GeoTIFF).

2. Формати даних

Формат даних (формат файлів) – це інформація за допомогою якої користувачі (як і операційна система) можуть швидко ідентифікувати вміст даних без необхідності зчитування вмісту всього файлу. Саме тому до вибору формату даних треба підходити відповідально.

Формати даних описують конкретну структуру для представлення даних у комп'ютері, що включає спосіб їх збереження, розмір пам'яті та правила кодування, часто у вигляді двійкових послідовностей.

Іноді формат оприлюднення даних викликає багато запитань і як наслідок – труднощів. Тож для кращого розуміння важливості коректного оприлюднення відкритих даних давайте звернемося до відомої класифікації «5 Stars Open Data», що була розроблена одним із творців Всесвітньої павутини Тімом Бернерсом-Лі.

Одна зірка – інформація доступна в мережі Інтернет у будь-якому форматі, але під відкритою ліцензією. У цю категорію потрапляють файли у форматі PDF, у тому числі скановані копії документів. Таку інформацію можна переглянути, роздрукувати та поширити, але опрацювати її без додаткових маніпуляцій (оцифрування) неможливо.

Дві зірки – дані оприлюднені у структурованому вигляді, проте формат даних не є відкритим (наприклад, XLSX). Багато користувачів для отримання даних залежать від комерційного програмного забезпечення. Такі дані можна обробляти автоматично, їх можна експортувати в інший формат, проте вони все ще містять зайві елементи оформлення, навігації, а значить також потребують додаткових дій для аналізу.

Три зірки – інформація доступна у відомих та добре описаних відкритих структурованих форматах (наприклад, CSV, JSON, XML). Користувачі можуть користуватися даними будь-яким чином та без необхідності використання комерційного програмного забезпечення. Проте з іншої сторони, така інформація все ще не є даними, що по-справжньому інтегровані у веб (in the Web).

Чотири зірки – ви використовуєте стандарти W3C21 (зокрема, RDF та SPARQL), ваші дані мають постійне посилання. Користувачі можуть отримати первинні набори відкритих даних у вигляді файлів (довідники, списки, таблиці у відкритому форматі, архів документів тощо) або через запит до API за вказаними параметрами. Це дає змогу отримувати тільки потрібну інформацію. Якщо у Вас є API – він має бути добре описаний, а доступ до нього може бути анонімний без обмежень або з реєстрацією, за вказаним ідентифікатором, лімітами на кількість одночасних запитів тощо.

П'ять зірок – ваші набори відкритих даних пов'язані між собою (linked data). Вони мають спільні довідники, класифікатори, ідентифікатори, посилання між документами та іншими елементами тощо. Дані являють собою семантичну мережу, що постійно оновлюється й змінюється відповідно до сучасних запитів. Від мережевого ефекту виграють і користувач, і публікатор.

Таким чином, відкритість та якість даних залежить не лише від форматів у яких здійснюється публікація, але й від способів доступу до інформації та кількості додаткових дій, які необхідні для її отримання, збереження та використання.

Найчастіше структуровані дані зустрічаються у відкритому форматі CSV. Також зустрічається формат XLS(X). Часто програмні та інформаційні системи дають змогу експортувати дані у форматах XML або JSON, тому їх часто використовують розпорядники для публікації набору даних. Втім, найкраще формати XML і JSON підходять для ієрархічних даних. Якщо ваші дані не є ієрархічними за природою, для їх публікації буде достатньо формату CSV. Для публікації структурованих (табличних) даних категорично не підходять формати: DOC(X), RTF, PDF, JPG, JPEG, TIFF, PNG.

TXT (Text File) – це стандартний текстовий формат, що містить текстові дані, які, як правило, організовані у вигляді рядків або взагалі не мають форматування. Даний формат є основою для багатьох більш спеціалізованих форматів, таких як HTML, XML, CSV. Файли TXT універсальні та вкрай прості за своєю природою, а тому їх можна відкривати на будь-якій операційній системі: за допомогою фактично будь-якого текстового редактора. У випадку Windows, наприклад, це Блокнот.

Крім того, даний формат дуже стійкий — кожне слово та символ у такому файлі самодостатні і, якщо трапиться пошкодження байтів, то зазвичай можна відновити дані за контекстом або продовжити обробку решти вмісту. В той же час, у стиснених чи двійкових файлах, пошкодження декількох байтів може зробити файл абсолютно невідновним. Головним недоліком використання файлів TXT є те, що вони займають більше місця у порівнянні з іншими файлами.

PDF (Portable Document Format) – універсальний файловий формат, який дозволяє зберігати шрифти, посилання, зображення, аудіо, відео, текст та власне сам макет вихідного документа незалежно від того, на якій із безлічі платформ чи додатків такий документ

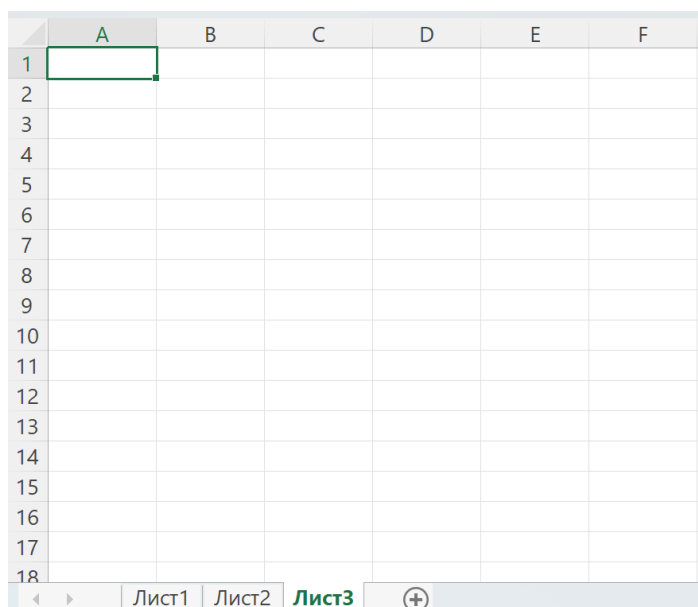
створювався. На сьогодні формат PDF, що був винайдений компанією Adobe, є відкритим стандартом, підтримуваним Міжнародною організацією зі стандартизації (ISO).

Беззаперечною перевагою формату PDF є той факт, що файли даного типу підтримують можливість електронного підпису і можуть переглядатися за допомогою безкоштовного програмного забезпечення Acrobat Reader DC.

3. Формати даних XLS та CSV

Формат даних *.xlsx* відомий нам передусім завдяки програмі Microsoft Excel. Звісно, крім відомого Excel, його можна прочитати й у більшості табличних редакторів: Apple Numbers, Google Spreadsheets, LibreOffice тощо.

Загалом XLSX-файл представляє собою віртуальну таблицю, розділену на стовпці та рядки, що формують комірки.



	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						

Рисунок 2.1 – Вигляд таблиці у програмі Microsoft Excel

У класичному табличному редакторі стовпці проіндексовані латинськими літерами, а рядки – цифрами. Кожна клітинка може містити як фіксовані дані, так і формули, часто пов'язані з даними, іноді навіть в інших файлах.

Формат XLSX дозволяє користувачеві змінювати форматування тексту: його шрифт, колір, вирівнювання (в комірці) і інші параметри. Крім того, документ може містити зображення, а також діаграми, побудовані на основі даних в певних комірках.

При роботі із даними форматом варто уникати об'єднаних комірок та пам'ятати, що колір та шрифт комірок у переважній більшості випадків не будуть нести ніякої інформації при машиночитній обробці даних.

CSV (від англ. comma-separated values, "значення, розділені комою") – простий текстовий відкритий формат, призначений для представлення табличних даних.

Формат CSV з'явився ще у 1970-х роках, коли комп'ютерні системи почали активно працювати з табличними даними. Хоча точна дата створення стандарту не зафіксована, CSV став фактичним стандартом зберігання та передачі структурованих даних задовго до появи формальних стандартів, таких як XML і JSON.

Формат CSV так і не був офіційно стандартизований. У 2005 році з'явився документ RFC 4180, який спробував описати загальні правила роботи з CSV, але цей документ не став обов'язковим стандартом. Через це досі існують різні варіації CSV: файли можуть використовувати кому, крапку з комою, табуляцію або інші символи як роздільники.

Файл у форматі CSV є звичайним текстовим документом, в якому дані організовані у вигляді рядків і стовпців. Кожен рядок відповідає одному запису, а значення всередині рядка поділяються спеціальними символами - роздільниками.

Найпростіший CSV-файл виглядає так:

```
name,category,price
Samsung A5,Smartphones,15000
iPhone 5S,Smartphones,14000
HP Laser Jet,Printers,12000
iMac 21.5,MonoBlocks,25000
```

У цьому прикладі перший рядок містить заголовки стовпців, а наступні рядки є даними.

Основні елементи структури CSV:

- Рядки: дані розділені рядково, кожен рядок - це новий запис;
- Розділювачі: найчастіше використовується кома, але в різних країнах можуть застосовуватись інші символи (точка з комою, табуляція та ін.);
- Заголовок: перший рядок може містити заголовки стовпців, але це не є обов'язковим.

Лапки та екранування символів – якщо одне із значень містить сам роздільник (наприклад, кома), його необхідно укласти в подвійні лапки. Якщо всередині значення зустрічаються лапки, вони повинні бути подвоєні.

Наприклад:

```
Name, Surname, Description
Ivan, Ivanov, "Developer, works in IT"
Ivan, Petrov, "Programmer, ""Senior"" specialist"
```

Залежно від регіону та налаштувань програмного забезпечення як роздільник у CSV-файлі можуть використовуватися:

- Кома (,) – стандартний роздільник в англomовних системах;
- Крапка з комою (;) – часто використовується в Європі, де кома застосовується як десятковий роздільник;
- Табуляція (\t) – використовується у форматі TSV (Tab-Separated Values);
- Пробіл () – зустрічається рідше, але можливий у деяких специфічних системах.

Однією з найпоширеніших проблем під час роботи з CSV є кодування символів. Якщо файл збережено в одному кодуванні, але відкривається в іншому, можливі помилки відображення (наприклад, "кракозябри" замість літер).

Найбільш популярні кодування:

- UTF-8 – універсальне кодування, рекомендоване для роботи з CSV.
- Windows-1251 – використовується в російськомовних версіях Windows.
- ISO-8859-1 – часто зустрічається у західноєвропейських системах.
- ANSI – застаріле кодування, яке може спричинити проблеми при переносі даних.

При збереженні файлів CSV у текстовому редакторі або таблицях (наприклад, Excel, Google Sheets) важливо вказувати кодування вручну. Найкращий варіант — UTF-8, оскільки він підтримує всі мови та символи.

Краще використовувати CSV в наступних випадках:

- Передача між системами – CSV-файли легко відкриваються у більшості програм, включаючи Excel, Google Sheets та бази даних;
- Зберігання невеликих та середніх обсягів даних – якщо дані займають до кількох мегабайт, CSV чудово впорається із завданням.

- Простота обробки – CSV можна легко читати та записувати у будь-якій мові програмування;
- Мінімальний розмір файлів – на відміну від XML або JSON CSV не містить зайвих тегів, що робить його компактним.

Не варто використовувати CSV в наступних випадках:

- Коли дані мають складну структуру – CSV не підтримує вкладені дані, на відміну від JSON та XML;
- Коли важлива цілісність даних – CSV не має вбудованої валідації даних, що робить його вразливим для помилок при імпорті;
- Коли потрібний швидкий доступ до даних – при роботі з великими файлами (сотні мегабайт) пошук CSV може бути повільним порівняно з базами даних;
- Коли дані містять символи-розділювачі – якщо значення містять коми, крапки з комою або лапки, CSV може стати важкочитаним і вимагати екранування.

4. Формати даних XML та JSON

На відміну від табличних форматів, формати JSON та XML слугують для збереження ієрархічних даних. Часто такі формати використовуються для передачі даних в інтернеті.

XML (Extensible Markup Language):

- Структура: XML використовує теги для визначення структури даних. Це робить його дуже гнучким, але також може бути громіздким.
- Читабельність: XML легко читається як людьми, так і машинами, але може бути складним для обробки через велику кількість тегів.
- Використання: Часто використовується в веб-сервісах, конфігураційних файлах та для обміну даними між різними системами

XML з'явився ще 1996 року і вперше був опублікований 1998 року. WWW Consortium (W3C) є розробником XML, і це стало Рекомендація W3C у 1998 році. XML виглядає дуже схожим на HTML, але XML є структурованішою мовою.

Документи у форматах HTML і XML містять дані, укладені в теги, але подібність між двома мовами закінчується. У форматі HTML теги *визначають оформлення даних* – розташування заголовків, початок абзацу тощо. У форматі XML теги *визначають структуру та зміст даних* – те, чим вони є.

Можна використовувати одну систему для генерації даних та позначки їх тегами у форматі XML, а потім обробляти ці дані в будь-яких інших системах незалежно від клієнтської платформи або операційної системи. Завдяки такій сумісності XML є основою однієї з найпопулярніших *технологій обміну даними*.

Правила XML дозволяють створювати будь-які теги, необхідні опису даних та його структури. Припустимо, що вам необхідно зберігати та спільно використовувати відомості про домашніх тварин. Для цього можна створити наступний XML-код:

```
<?xml version="1.0"?>
<CAT>
  <NAME>Izzy</NAME>
  <BREED>Siamese</BREED>
  <AGE>6</AGE>
  <OWNER>Colin Wilcox</OWNER>
</CAT>
```

За тегами XML зрозуміло, які дані ви переглядаєте. Наприклад, ясно, що це дані про kota, і можна легко визначити його ім'я, вік тощо. Завдяки можливості створювати теги, що визначають майже будь-яку структуру даних, мова XML розширюється.

Правильно сформований XML-файл має відповідати дуже строгим правилам. Якщо він не відповідає цим правилам, не працює XML.

Наприклад, у попередньому прикладі кожен тег, що відкриває, має відповідний тег, що закриває, тому в даному прикладі дотримано одне з правил правильно сформованого XML-файлу.

Базовий синтаксис XML:

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Декларація (оголошення) XML містить відомості, які готують процесор XML до синтаксичного аналізу документа XML.

Наступний синтаксис показує оголошення XML:

```
<?xml
  version = "version_number"
  encoding = "encoding_declaration"
  standalone = "standalone_status"
?>
```

Декларація XML складається з версії XML, кодування символів та/або автономного статусу. Декларація є необов'язковою. *Якщо декларація XML присутня, вона має бути першою.*

Таблиця 2.1 Атрибути та їх значення в декларації XML

Параметр	Parameter_value	Parameter_description
Версія	1.0	Задає версію стандарту XML.
Кодування	UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 до ISO-8859-9, ISO-2022-JP, Shift_JIS, EUC-JP	Він визначає кодування символів у документі. UTF-8 – кодування за замовчуванням.
Автономний	Так чи ні	Він повідомляє синтаксичному аналізатору, чи залежить документ від інформації із зовнішнього джерела, такого як визначення типу зовнішнього документа (DTD) для свого вмісту. Значення за замовчуванням – ні. Встановлення цього значення уєс повідомляє процесору, що для синтаксичного аналізу документа не потрібно ніяких зовнішніх оголошень.

Наприклад:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

UTF-8 використовує 8 біт для представлення символів. UTF-16 для представлення символів використовує 16 біт.

Оголошення типу XML-документа, широко відоме як DTD, дозволяє точно описати мову XML. DTD перевіряють словниковий запас та правильність структури XML-документів на відповідність граматичним правилам відповідної мови XML. XML DTD може бути вказаний усередині документа або може зберігатися в окремому документі.

Приклад внутрішнього DTD:

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address [
    <!ELEMENT address (name,company,phone)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT company (#PCDATA)>
    <!ELEMENT phone (#PCDATA)>
]>

<address>
    <name> ... </name>
    <company> ... </company>
    <phone> ... </phone>
</address>
```

Коментарі не є обов'язковими. Додавання коментарів допомагає зрозуміти зміст документа. Починається з <!-- і закінчується -->.

Наприклад:

```
<!-- Add your comment here -->
```

Теги працюють парами, крім оголошень. Кожна пара тегів складається з тега, що відкриває (початковий тег) і тега, що закриває (кінцевий тег). Імена тегів розташовуються у <>. Для конкретної пари тегів початковий та кінцевий теги мають бути ідентичними, за винятком того, що кінцевий тег має / після <.

Наприклад:

```
<name>...</name>
```

Теги чутливі до регістру. Приклад невірного тегу:

```
<name>...</Name>
```

Все, що знаходиться між тегами, що відкриває і закриває, називається *зміст*.

Відкриваючий тег, контент і тег, що закриває, в цілому називається *елемент*.

Елементи можуть містити *атрибути*.

Наприклад маємо:

```
<age>20</age>
```

age – це ім'я елемента (або елемент); 20 – зміст;

Якщо між тегами немає контенту, це називається *порожні теги*.

Наприклад:

```
<result></result>
або
<result/>
```

XML – документ повинен містити рівно один кореневий елемент.

Теги мають бути розташовані строго одне всередині одного.

Наприклад:

```
<b><u>This text is bold and italic</u></b>
```

Атрибут елемента розміщується після імені тега у початковому тегу. Можна додати більше одного атрибута до одного елемента з різними іменами атрибутів. Значення атрибутів мають бути поміщені в лапки. Елемент не може містити декілька атрибутів з однаковим назвою.

Атрибут XML має наступний синтаксис

```
<element-name attribute1 attribute2 > ....content.. < /element-name>
```

де attribute1 та attribute2 має наступну форму:

```
name = "value"
```

Наприклад:

```
<teacher subject="English">
  <name>Mr. John</name>.
  <qualification>Graduate </qualification>
</teacher>
```

Ще один приклад:

```
<garden>
  <plants category = "flowers" />
  <plants category = "shrubs"> </plants>
</garden>
```

XML-сутності – це спосіб представлення спеціальних символів. Деякі символи (наприклад, & та < тощо) зарезервовані в XML. Їх називають *спеціальні символи* і вони не можуть бути використані безпосередньо для інших цілей.

Символ	Опис	Ім'я сутності
"	Кавичка (double цитата)	"
&	Амперсанти	&
'	Апостроф (одинарна лапка)	'
<	Знак менший	<
>	Знак більше	>

Наприклад:

```
<friend>
  <name>My friends are Alice &amp; Jane</name>
</friend>
```

На додаток до правильно сформованих даних з тегами XML-системи зазвичай використовують два додаткові компоненти:

– схеми;

–перетворення.

Схема XML відома як *XML Schema Definition (XSD)*. Він використовується для опису та перевірки структури та змісту XML-даних. Схема XML визначає елементи, атрибути та типи даних. Елемент схеми підтримує простір імен. Це схоже на схему бази даних, яка описує дані у базі даних.

Основна ідея схем XML полягає в тому, що вони описують допустимий формат, який може приймати документ XML.

Приклад схеми:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
  <xs:element name = "contact">
    <xs:complexType>
      <xs:sequence>
        <xs:element name = "name" type = "xs:string" />
        <xs:element name = "company" type = "xs:string" />
        <xs:element name = "phone" type = "xs:int" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Обумовлені прості типи:

xs: integer, xs: boolean, xs: string, xs: date.

Складний тип – це контейнер для інших визначень елементів. Це дозволяє вказати, які дочірні елементи може містити елемент, та надати деяку структуру:

```
<xs:element name = "Address">
  <xs:complexType>
  </xs:complexType>
</xs:element>
```

Для перевірки XML-файлів можна скористатися онлайн-додатком, наприклад <https://www.freeformatter.com/xml-validator-xsd.html>.

XML дозволяє ефективно використовувати та повторно використовувати дані. Механізм повторного використання даних називається *перетворенням XSLT* (або просто перетворенням).

Припустимо, що в базі даних А дані про продаж зберігаються в таблиці, зручній для відділу продажу. У базі даних Б зберігаються дані про доходи та витрати у таблиці, спеціально розробленої для бухгалтерії. База Б може використовувати перетворення, щоб прийняти дані від бази даних А і помістити їх у відповідні таблиці.

Поеднання файлу даних, схеми та перетворення утворює базову систему XML. Файл даних перевіряється на відповідність до правил схеми, а потім передається будь-яким придатним способом для перетворення.

XSLT найчастіше використовується для перетворення даних між різними схемами XML або перетворення даних XML на веб-сторінки або документи PDF.

JSON (JavaScript Object Notation) – це простий, популярний і легкий для читання формат обміну даними. Він був розроблений з метою забезпечення зручного способу передачі інформації між додатками.

Історія JSON бере свій початок 2001 року, коли Дуглас Крокфорд представив цей формат, щоб полегшити роботу з даними в Аїах-додатках.

Структура JSON – вся інформація зберігається у вигляді *пар ключ-значення*.

Ключі в JSON це рядки, укладені в подвійні лапки. Ключі відіграють роль ідентифікаторів для значень і допомагають структурувати дані. Вони є унікальними в межах одного об'єкта JSON, що означає, що кожен ключ в об'єкті має бути унікальним.

Значення в JSON можуть бути різними типами даних, включно з рядками (текст), числами, логічними значеннями (true або false), null (порожнє значення), масивами або вкладеними об'єктами. Значення можуть бути укладені в подвійні лапки (для рядків), без лапок (для чисел, логічних значень і null) або в квадратні дужки чи фігурні дужки (для масивів і об'єктів відповідно).

Приклад JSON-файлу:

```
{
    "name": "John",          // Строкове значення
    "age": 30,               // Числове значення
    "isStudent": true        // Логічне значення
    "address": null          // значення null (пусто)
}
```

де name, age, isStudent та address – це ключі, а їхні значення відповідно «John», 30 і true.

Масиви в JSON являють собою впорядковані списки елементів. Вони укладаються в квадратні дужки і можуть містити значення різних типів даних, включно з іншими масивами та об'єктами.

Наприклад:

```
{
    "fruits": ["apple", "banana", "orange"]
}
```

де fruits – це ключ, а його значення – масив з елементами apple, banana і orange.

Об'єкти являють собою невпорядковані набори ключів і значень. Вони містяться у фігурних дужках і дають змогу структурувати дані складніше, вкладаючи об'єкти один в одного.

Наприклад:

```
{
    "person":
    {
        "name": "Alice",
        "age": 25
    }
}
```

person – ключ, а його значення являє собою вкладений об'єкт із ключами name і age.

Серіалізація і десеріалізація – це два важливі процеси, пов'язані з JSON, які дають змогу перетворити дані у формат JSON і назад.

Серіалізація JSON – це процес перетворення даних з об'єктів і структур даних у рядок JSON. Коли ми серіалізуємо дані, ми робимо їх придатними для передавання мережею або збереження на диску в текстовому форматі.

Десеріалізація JSON – це зворотний процес, під час якого рядок JSON перетворюється назад в об'єкти і структури даних. Коли ми отримуємо дані у форматі JSON з мережі або файлу, нам потрібно перетворити їх назад на об'єкти, щоб використовувати їх у програмі.

Запитання для самоперевірки

1. Які типи даних за змістом та призначенням ви знаєте?
2. Що таке формати даних?
3. Що представляє собою формат даних XLS?
4. Що представляє собою формат даних CSV?
5. Як виглядає CSV-файл?
6. Що представляє собою формат даних XML?
7. Як виглядає XML-файл?
8. Які є правила при створенні XML-файлу?
9. Що представляє собою формат даних JSON?
10. Як виглядає JSON-файл?
11. Що таке серіалізація та десеріалізація?

Лекція 3. Структури даних

1. Основні визначення
2. Масиви і робота з ними
3. Зв'язний список

1. Основні визначення

Дані – це великий термін, що охоплює всі типи інформації, до найпростіших чисел і рядків.

У класичній програмі Hello World! рядок "Hello World!" Буде фрагментом даних. По суті, навіть найскладніші фрагменти даних зазвичай складаються з чисел і рядків.

Структура даних – це те, як дані організовані. Одні й самі дані можуть бути організовані по-різному.

Структури даних – це засоби організації та зберігання інформації в комп'ютерних програмах, що дозволяє ефективно управляти даними. Вони допомагають логічно структурувати та групувати інформацію, роблячи код більш зрозумілим та підтримуваним.

Структури даних можуть бути організовані у певному форматі, що дозволяє розробникам ефективно обробляти, зберігати та передавати дані.

Організація даних важлива не просто сама по собі, вона може значно вплинути на швидкість виконання вашого коду. Залежно від обраного способу організації даних ваша програма може працювати швидше або повільніше, причому різниця може бути в кілька порядків.

А якщо ви створюєте програму, якій потрібно обробляти великі обсяги даних, або веб-додаток, що використовується одночасно тисячами людей, то від вибраних структур даних може залежати, чи буде ваше ПЗ нормально працювати або збоїти через надмірне навантаження.

Є чотири основні способи використання структур даних, які називають операціями:

- Читання – одержання елемента з певного місця структури даних;
- Пошук – знаходження певного значення у структурі даних;
- Вставка – додавання нового значення до структури даних;
- Вилучення – виключення значення структури даних.

При вимірі швидкості виконання операції ми враховуємо не кількість часу, а кількість кроків, які потрібно. Вимірювати швидкість операції в секундах чи хвилинах ненадійно, оскільки тривалість залежить від обладнання, у якому вона виконується. Вимір швидкості виконання операції по-іншому ще називають *виміром її тимчасової складності*.

Терміни швидкість, тимчасова складність, ефективність, продуктивність та час виконання є синонімами. Усі вони відносяться до кількості кроків, необхідних виконання операції.

2. Масиви і робота з ними

Масив – це велика кількість неперервної області пам'яті, доступ до якої здійснюється за допомогою індексів (їх кількість вказує на розмірність масиву). Його легко уявити у вигляді сукупності комірок, в яких програма зберігає значення.

Масив складається з елементів одного типу, званого базовим, тому структура масиву однорідна. Базовий тип, у свою чергу, може бути як скалярним, так і структурованим або абстрактним, тобто елементами масиву можуть бути числа, символи, рядки, структури, списки, дерева, у тому числі і інші масиви.

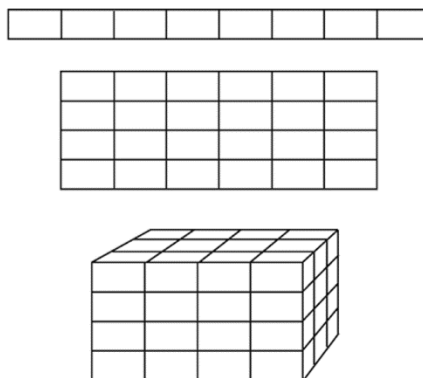


Рисунок 3.1 – Приклади масивів

Число елементів масиву фіксовано, тому обсяг займаної масивом пам'яті залишається незмінним.

З кожним елементом масиву пов'язаний один (для одновимірного масиву) або декілька індексів (для багатовимірного масиву). Вони однозначно визначають місце елемента в масиві і забезпечують прямий та швидкий доступ до цього елемента.

Індекс масиву – це число, що визначає позицію елемента масиву. У більшості мов програмування нумерація індексів починається з нуля.

Індекси масиву відносяться до певного порядкового типу, тому індекси можна обчислювати. Це забезпечує, з одного боку, гнучкість обробки елементів масиву, з іншого боку, створює небезпеку виходу за межі масиву, якщо не передбачені відповідні засоби контролю.

Областями застосування масивів є:

- числові масиви в обчислювальних задачах;
- матрична алгебра;
- таблиці;
- керуючі та інформаційні таблиці в операційних системах, трансляторах, системах управління базами даних (СУБД);
- представлення інших структур: графів, дерев, тощо.

Вся інформація, необхідна для управління масивом, задається при його оголошенні в програмі. Оголошення містить ім'я масиву, тип елементів, який однозначно визначає довжину елемента, діапазони зміни індексів або число значень індексів, якщо нижні межі індексів фіксовані.

Таким чином, загальна кількість елементів масиву і розмір пам'яті для масиву повністю визначаються при оголошенні масиву.

Транслятор виділяє необхідну кількість пам'яті і будує керуючий блок масиву - дескриптор, або інформаційний вектор, наприклад, такого змісту (для одновимірного масиву):

- адреса початку масиву A_n ;
- тип елемента (довжина елемента масиву L);
- нижня границя індекса i_n ;
- верхня границя індекса i_v .

У разі багатовимірного масиву для кожного індексу задається значення нижньої і верхньої межі або ж для кожного рядка (кожного індексу) створюється свій дескриптор.

Цієї інформації достатньо як для доступу до елементів масиву, так і для контролю над тим, щоб значення індексів не виходили за встановлені діапазони.

Доступ до будь-якого i -го елемента в одновимірному масиві здійснюється за адресою:

$$A_i = A_0 + i * L$$

де A_0 – початок масиву, тобто адреса нульового елемента; i – індекс елемента; L – розмір елемента в байтах.

Вільними називають двовимірні масиви, розмір кожного рядка в яких може бути різним. Оскільки стандарти мови програмування не допускають такого виду масивів, то створюється симбіоз масивів: одновимірний масив покажчиків, число елементів n якого дорівнює числу рядків змінної довжини, і n одновимірних масивів різної довжини.

Таким чином, масив покажчиків має фіксовану довжину, значить, фіксоване і число рядків вільного масиву. А вже пам'ять під кожен рядок вільного масиву виділяється динамічно.

На рисунку нижче зображена структура зберігання вільного масиву з n рядками. Масив покажчиків U має n елементів, кожен з яких містить адреси векторів – рядків масиву.

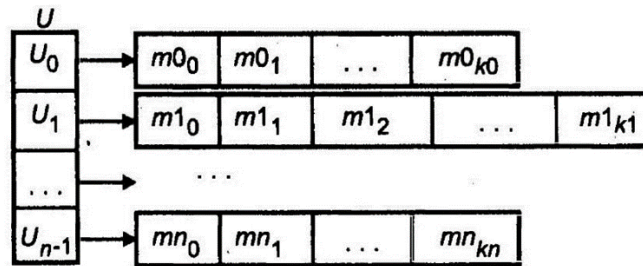


Рисунок 3.2 – Структура зберігання вільного масиву з n рядками

Для управління вільним масивом може створюватися дескриптор, в найпростішому випадку – це покажчик на масив покажчиків. Крім того, створюється допоміжний лінійний масив, який буде зберігати відповідні довжини рядків. За аналогічною схемою можуть бути створені і двовимірні масиви з однаковими довжинами рядків. Вільні масиви часто використовуються для представлення інших, більш складних структур.

До недоліків можна віднести потребу в додатковій пам'яті під масив покажчиків і розмірів рядків та багаторазове звернення до "купи" для отримання пам'яті під рядки.

До переваг можна віднести наступне:

- при роботі з дуже великими масивами отримання великих суцільних областей може стати неможливим; в той же час отримання пам'яті під кожен рядок змінного розміру цілком можливе;
- з'являється можливість обробляти великі масиви даних, що зберігаються в файлах, розміщуючи в оперативній пам'яті тільки ті рядки, які спільно обробляються в даний момент;
- застосування вільних масивів, що мають рядки різної довжини, дозволяє більш ефективно використовувати пам'ять;
- при сортуванні масиву по рядках (не елементів в рядку, а самих рядків) замість переставляння рядків в пам'яті можна переставляти тільки елементи масиву покажчиків, в той час як самі рядки залишаються на місці.

Розглянемо трикутні масиви.

Розглянемо, наприклад, матрицю суміжності, яка представляє зв'язки між вузлами мережі. В якості останньої можуть виступати літаки, що літають між різними аеропортами. Якщо відбувається політ від аеропорту i до аеропорту j , то елемент масиву $connected[i, j]$ приймає значення 1.

Припустимо також, що існує зворотний маршрут від аеропорту j до аеропорту i , це дає нам елемент $connected[j, i] = connected[i, j]$. Виходить, що немає необхідності зберігати $connected[i, j]$ і $connected[j, i]$ одночасно, оскільки вони однакові. І отже, програма може заощадити пам'ять, перетворивши матрицю суміжності в трикутний масив.

У трикутній матриці $A_{n \times n}$ всі елементи вище або нижче головної діагоналі є нульовими. Якщо нульовими є елементи вище головної діагоналі (нижня трикутна матриця), то ненульовими будуть елементи $a_{11}; a_{21}, a_{22}; a_{31}, a_{32}, a_{33}; \dots; a_{n1}, a_{n2}, \dots, a_{nn}$.

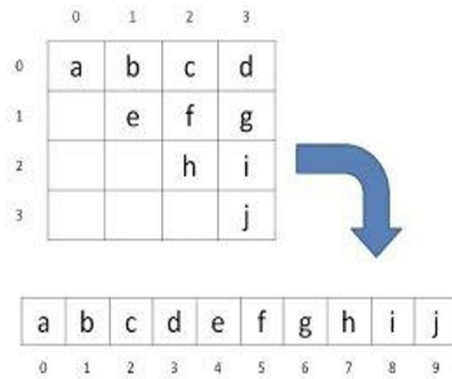


Рисунок 3.3 – Приклад трикутного масиву

Розрідженими називаються матриці, що містять велику кількість нульових елементів. Подання таких матриць у вигляді двовимірних масивів призводить до нераціонального використання пам'яті і неефективності виконуваних над ними операцій. Тому, для зберігання таких матриць використовуються спеціальні схеми.

Перший і найбільш простий спосіб полягає в наступному. Створюється двовимірний масив розміром $(n \times 3)$, де n – число ненульових елементів вихідної матриці. Перші два елементи кожного рядка містять індекси рядка і стовпця ненульового елемента, а третій – сам ненульовий елемент.

$$A = \begin{vmatrix} 0 & 0 & 7 & 0 & 8 \\ 2 & 0 & 0 & 6 & 9 \\ 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \end{vmatrix}$$

	STR	STL	ZN
1	3	7	
1	5	8	
2	1	2	
2	4	6	
2	5	9	
3	1	4	
4	3	5	

Рисунок 3.4 – Приклад подання розрідженої матриці

3. Зв'язний список

Зв'язний список являє собою структуру даних, яка вільна від обмежень у масивах.

Зв'язний список не зберігає свої елементи в послідовних комітках пам'яті, і користувач може як додати будь-яку кількість нових елементів до нього в довільний момент, так і видалити елементи, які стали непотрібними зі списку без необхідності виділення нових зв'язних областей в "купі" і перезапису списку на нове місце.

Проте, на відміну від масиву, зв'язний список не допускає прямого (випадкового) доступу до даних. Елементи в зв'язному списку можуть бути доступні тільки в послідовному порядку. Але на відміну від масиву, вставка і видалення елемента списку може бути виконана в будь-який момент в реальному часі.

Зв'язаний список, являє собою лінійну сукупність елементів даних, які називаються *вузлами* і так званий *дескриптор* (або покажчик на початок) списку.

Кожен з вузлів містить покажчик (адресу) наступного вузла в списку. Останній вузол у списку містить NULL-покажчик, який і сигналізує про кінець списку.

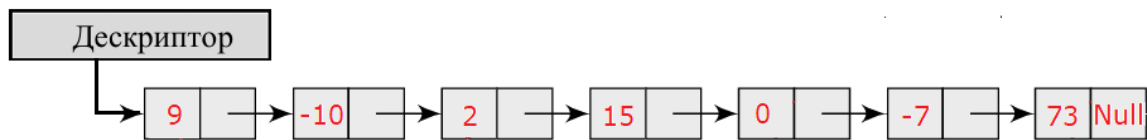


Рисунок 3.5 – Приклад зв'язного списку

Фізично список реалізується як сукупність дескриптора списку і елементів, розміщених в оперативній пам'яті або на зовнішніх запам'ятовуючих пристроях (ЗЗП) і зв'язаних один з одним у ланцюжок за допомогою показників.

Дескриптор зазвичай реалізується у вигляді структури і може містити наступну інформацію про список:

- тип структури;
- назву списку;
- показник (адреса) початку списку;
- поточне число елементів;
- опис елемента (довжина, тип, тощо).

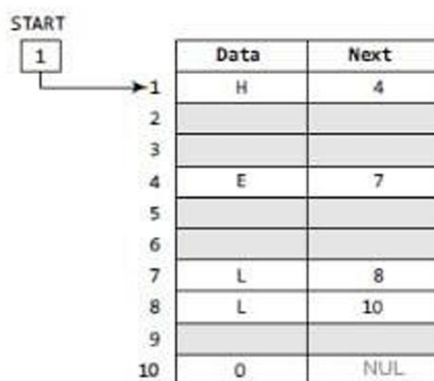
Зв'язаний список є абстрактною структурою даних, яка в свою чергу, часто використовується для реалізації інших, більш складних абстрактних структур даних. Таким чином, він виступає в якості будівельного блоку для реалізації структур даних, таких як стеки, черги і їх варіації.

Зв'язний список може сприйматися як поїзд або послідовність вузлів, в яких кожен вузол містить один або кілька полів даних і показник на наступний вузол. Ліва частина вузла, яка і містить дані, може включати в себе простий тип даних, масив або структуру. Права частина вузла містить показник на наступний вузол (або адресу наступного вузла в послідовності). Останній вузол не матиме за собою ніякого наступного вузла, так що він буде зберігати спеціальне значення, зване NULL.

Зв'язні списки містять дескриптор, який зберігає адресу першого вузла в списку. Ми можемо обійти весь список за допомогою дескриптора, який містить адресу першого вузла; адресна частина першого вузла в свою чергу, зберігає адресу наступного за ним вузла. Використовуючи цю техніку, окремі вузли списку утворюють ланцюжок вузлів

Якщо дескриптор в якості адреси першого вузла містить NULL, то зв'язний список порожній і не містить вузлів.

Наприклад:



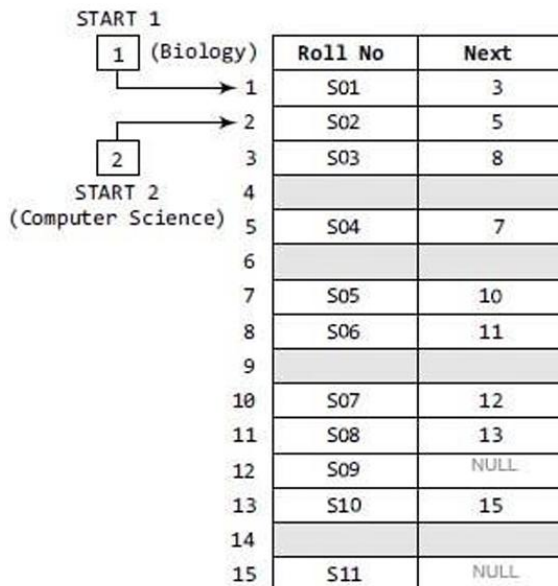
Дескриптор START вказує перший елемент зв'язного списку в пам'яті.

В прикладі START = 1, так що перший вузол списку зберігається за адресою 1. Перший вузол містить дані data = 'H', які є символ 'H'. Відповідний next зберігає адресу наступного вузла, що дорівнює 4. Таким чином, ми будемо бачити адресу 4, щоб витягти наступний елемент даних, тощо. Ми повторюємо цю процедуру до тих пір, поки не досягнемо позиції, де наступний вузол в поле next містити NULL, так як це означає кінець зв'язного списку.

Зверніть увагу, що в прикладі показується блок пам'яті, з адресами від 1 до 10. Заштрихована частина містить дані інших застосувань.

Вузли зв'язного списку не обов'язково повинні бути в зв'язаних комірках пам'яті. У нашому прикладі вузли для зв'язного списку, зберігаються за адресами 1, 4, 7, 8 і 10.

Приклад, два зв'язних списки одночасно розташовані в одній області пам'яті:



В прикладі вище, учні школи хочуть вибрати вивчення предметів: біологію або інформатику. Тепер у нас буде два зв'язаних списки, по одному для кожного предмета. Кожен із списків буде містити номери всіх учнів, які вибрали для вивчення відповідний предмет.

Односпрямований список є найпростішим типом зв'язного списку, в якому кожен вузол містить деякі дані і показчик на наступний вузол того ж списку. Говорячи про те, що вузол містить показчик на наступний, ми маємо на увазі, що вузол зберігає адресу наступного вузла в послідовності. Односпрямований список дозволяє виконувати обхід даних тільки в один бік.

Зв'язний список складається з вузлів, які розділені на дві частини: інформаційну і адресну.

Таким чином, пошук призначений для виявлення того, чи присутнє в інформаційній частині вузлів списку задане значення, чи ні. Якщо воно присутнє, то алгоритм повертає адресу вузла, який містить таке значення.

Новий вузол додається у вже існуючий зв'язний список. Розглянемо чотири випадки такої вставки.

- Випадок 1: Новий вузол вставляється у верхню частину (початок) списку.
- Випадок 2: Новий вузол вставляється в кінець списку.
- Випадок 3: Новий вузол вставляється після заданого вузла.
- Випадок 4: Новий вузол вставляється Випадок 3: Видаляється вузол після заданого вузла.
- перед заданим вузлом.

Вузол видаляється з уже існуючого зв'язного списку. Ми розглянемо три випадки, і подивимося, як виконується видалення у кожному конкретному випадку.

- Випадок 1: Видаляється перший вузол списку.
- Випадок 2: Видаляється останній вузол списку.

Спустошення списку (Underflow List) виникає тоді, коли ми намагаємося видалити вузол із зв'язного списку, який вже порожній.

Коли ми видаляємо вузол із зв'язного списку, ми насправді повинні звільнити пам'ять, зайняту цим вузлом і повернути її в пул вільних адрес "купи" для використання в подальшому.

Двоспрямований список є більш складним типом зв'язного списку, який містить в кожному вузлі два показники: *показчик на наступний вузол* і *показчик на попередній вузол* в ланцюжку вузлів. Таким чином, вузол складається щонайменше з трьох частин: інформаційної, показника на наступний і показника на попередній вузол.

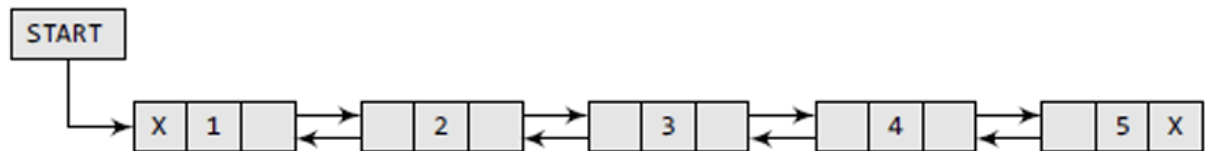


Рисунок 3.6 – Приклад двоспрямованого списку

Двоспрямований список вимагає більше місця в пам'яті під вузол і більш складних базових операцій маніпулювання списком. Проте, двоспрямований список забезпечує легкість маніпулювання елементами списку, оскільки він підтримує показники на вузли в обох напрямках (вперед і назад).

Наявність двох показників замість одного надає кілька переваг. Найбільш важлива з них полягає в тому, що переміщення по списку стає можливим в обох напрямках. Це спрощує роботу зі списком, зокрема, вставку і видалення вузлів. Крім цього, користувач може переглядати список в будь-якому напрямку.

Ще одна перевага має значення тільки при деяких несправностях. Оскільки весь список можна пройти не лише за прямим, але і за зворотним посиланням, то в разі, якщо якийсь із показників стане невірним, цілісність списку можна відновити за іншим ланцюгом посилань.

У *циклічному зв'язному списку*, останній вузол містить показник на перший вузол списку. Використовуються як циклічний односпрямований, так і циклічний двоспрямований списки. Здійснюючи навігацію по такому списку, ми можемо почати в будь-якому вузлі і проходити по списку в будь-якому напрямку, вперед або назад, поки ми не досягнемо того ж вузла, з якого ми розпочали.

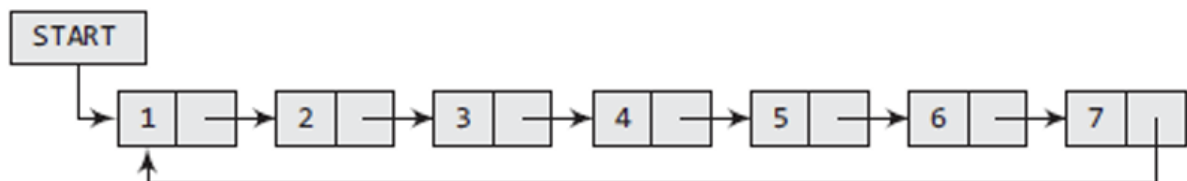


Рисунок 3.7 – Приклад циклічного односпрямованого списку

Таким чином, циклічний зв'язний список не має ні початку, ні кінця. Умовно, початком циклічного списку вважається вузол за адресою, вказаної в дескрипторі списку.

Єдиним недоліком циклічного зв'язного списку є складність навігації. Справа в тому, що в циклічному списку відсутній останній вузол і, отже, немає значення NULL в адресній частині будь-якого з вузлів списку. Це ускладнює індикацію моменту завершення перегляду списку.

Циклічні списки широко використовуються в багатозадачних ОС для обслуговування задач. Так, ОС здатна повторювати цикл завдань, щоб по черзі запускати і зупиняти кожну з них, виділяючи задане число квантів часу процесора для її виконання. Якщо почався новий процес, він може бути доданий в будь-яке місце списку, наприклад за дескриптором, що дозволить йому відразу ж і запуснитися.

Ще один приклад використання циклічного списку – це Інтернет. Коли ми працюємо в Інтернеті, ми можемо використовувати кнопку Назад і кнопку Вперед браузера для переходу до попередніх / наступних сторінок. Для цього структура циклічного списку використовується в браузері для підтримки послідовності відвіданих Web сторінок.

В односпрямованому зв'язному списку кожен вузол містить посилання лише на наступний елемент, в двоспрямованому – на наступний і попередній. Це необхідно для того, щоб забезпечити два способи переміщення – прямий і зворотний. Однак до структури вузлів списку можна додавати і інші посилання, щоб передбачити додаткові способи переміщення і в інших напрямках.

Припустимо, ми сконструювали структуру Planet для зберігання інформації про планети Сонячної системи і призначили їй поле з ім'ям Distance, в якому містяться дані про відстань від планети до Сонця. З огляду на це список буде організований (відсортований) у наступному порядку: Меркурій, Венера, Земля, Марс, Юпітер, Сатурн, Уран, Нептун (і Плутон, при бажанні). Аналогічним чином можна додати і інші поля, щоб впорядкувати планети по їх масі (поле weight), діаметру (поле D), тощо.

Кожен ланцюжок вузлів списку, відсортований за новим полем, буде визначатися новим набором адресних показників списку і буде називатися потоком.

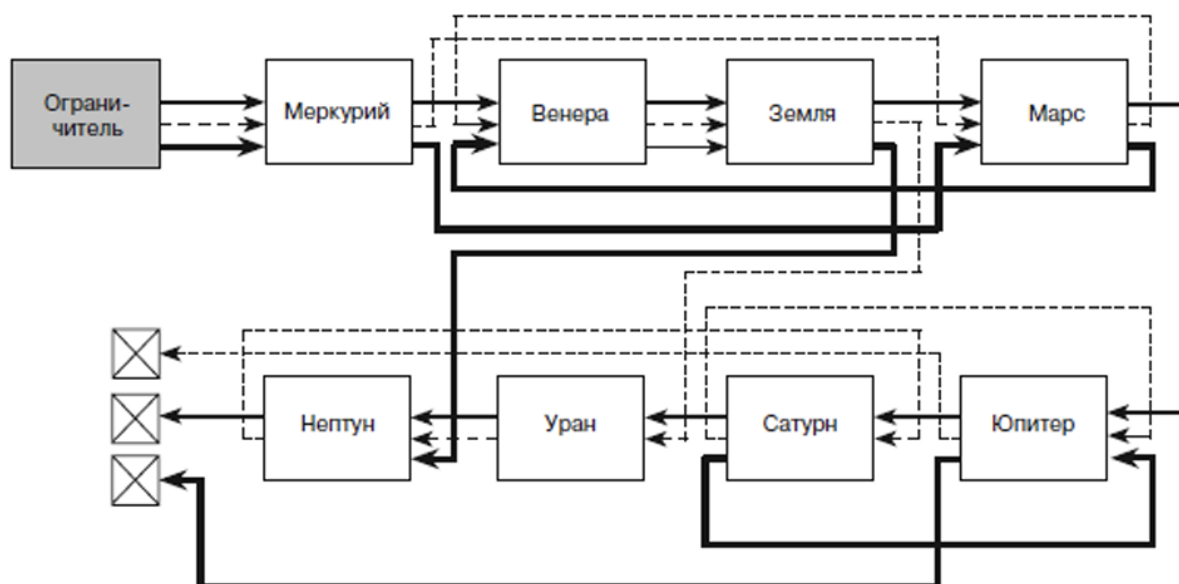


Рисунок 3.8 – Приклад багатозв'язного односпрямованого списку

В прикладі показаний зв'язний список планет з трьома потоками. Посилання, позначені тонкою лінією, вказують на потік сортування по віддаленості від Сонця, пунктирною – на потік по масі, а товстої – на потік по діаметру.

Таким чином, можна бачити, що в багатозв'язних списках, кожен вузол може мати n чи $2n$ показників на інші вузли, де n – число потоків списку. двоспрямований список є окремим випадком багатозв'язного списку при $n=1$. Однак, на відміну від двоспрямованого списку, вузли в багатозв'язних списках можуть бути чи не бути двоспрямованими окремо для кожного з потоків.

Запитання для самоперевірки

1. Що таке структури даних та для чого вони потрібні?
2. Які є основні способи використання структур даних?
3. Що таке масиви та як з ними працювати?
4. Які є області застосування масивів?

5. Що таке вільний масив?
6. Що таке розріджені матриці?
7. Що таке зв'язний список та як з ним працювати?
8. Що таке дескриптор та для чого він використовується?
9. Яка структура односпрямованого списку?
10. Яка структура двоспрямованого списку?
11. Яка структура циклічного зв'язного списку?

Лекція 4. Структури даних (Продовження)

1. Стеки
2. Черги

1. Стеки

Стеки і черги – відносно прості структури даних, які зберігають об'єкти в порядку «першим прийшов, першим вийшов» або «останнім прийшов, першим вийшов». По мірі необхідності їх можна розширювати, щоб додавати додаткові елементи, чи скорочувати для видалення елементів, подібно зв'язним спискам.

Для створення стеків і черг найчастіше використовуються зв'язні списки, хоча можливе використання й масивів.

Стеки і черги ідеально підходять для моделювання реальних ситуацій – каналів обслуговування в банку, супермаркеті або операційній системі. Однак найчастіше в них зберігають об'єкти, які будуть в подальшому оброблятися за допомогою інших алгоритмів, наприклад мережевих алгоритмів найкоротшого шляху.

Стек – це структура даних, в якій елементи додаються і видаляються у порядку «останнім прийшов, першим пішов». Через таку поведінку їх іноді називають списками LIFO, або просто LIFO (від англ. Last in first out).

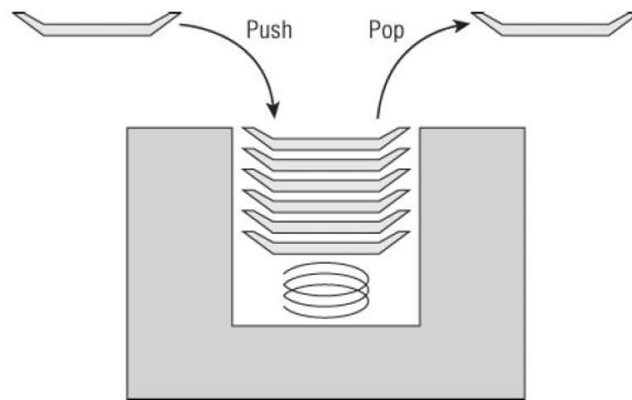


Рисунок 4.1 – Приклад структури стека

Стек легко уявити у вигляді стопки книг, що лежить на столі: можна додати або прибрати книгу зверху, але не можна вийняти її з середини або знизу, не зруйнувавши всю конструкцію.

Таким чином, стек являє собою динамічно змінюваний набір елементів, в якому запис і вибірка елементів проводиться лише з одного кінця. Тобто вибірка елементів із стека завжди здійснюється в порядку, зворотному їх запису.

Області застосування стека:

- реалізація механізму виклику функцій і передача їм параметрів;
- трансляція (синтаксичний і семантичний аналізи, генерація кодів тощо);
- реалізація механізму рекурсії;
- реалізація управління динамічною пам'яттю, тощо.

Стеки можуть бути представлені в пам'яті або у вигляді масиву (вектора), або у вигляді зв'язного списку.

При векторному поданні під стек відводиться суцільна область пам'яті, досить велика, щоб в ній можна було помістити максимальне число елементів, що визначається розв'язуванням завданням.

Граничні адреси цієї області є параметрами фізичної структури стека – вектора. В процесі заповнення стека місце останнього введенного елемента (його адреса) знаходиться в

покажчику вершини стека. Якщо показчик вийде за верхню межу стека, то стек вважається переповненим і включення нового елемента стає неможливим.

Тому для векторного стека треба відводити досить велику пам'ять, розмір якої визначається максимально можливою ємністю стека. Проте якщо векторний стек в процесі своєї роботи заповнюється тільки частково, то пам'ять використовується неефективно.

Так як під стек відводиться фіксований обсяг пам'яті, а кількість елементів усе-ж таки змінна, то кажуть, що стек в векторній пам'яті - це напівстатична структура даних. Зазвичай в стеку елементи мають один і той самий тип, тому обробка такого стека досить проста.

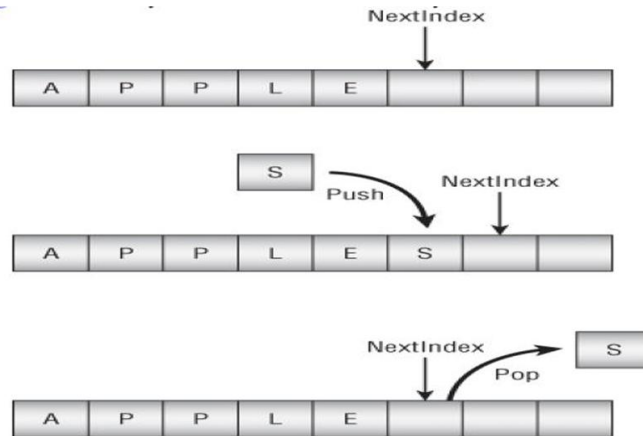


Рисунок 4.2 – Стек, представлений як вектор

Другий спосіб реалізації стека – це реалізація його структури за допомогою зв'язного списку. При списковому представленні стека пам'ять під дескриптор і під кожен елемент стека виділяється динамічне; включення і вибірка елемента здійснюються на початку списку, який є одночасно і вершиною стека.

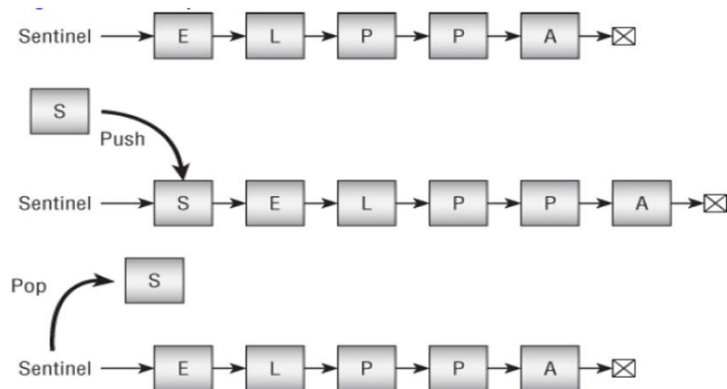


Рисунок 4.3 – Представлення стека у вигляді зв'язного списку

Переповнення стека в цьому випадку не відбувається, або відбувається досить рідко (лише при переповненні самої області динамічної пам'яті програми), проте алгоритми обробки складніші, а час обробки збільшується, оскільки операції включення та вибірки елементів пов'язані зі зверненням до операційної системи для отримання або звільнення пам'яті.

Векторний стек не потребує додаткової пам'яті для зберігання посилань між елементами, але в ньому нераціонально використовується місце під елементи.

Розмір відведеної пам'яті під векторний стек залежить від програми і від того, чи відома заздалегідь максимальна кількість елементів. Якщо такої інформації немає – при необхідності можна змінити розмір динамічного масиву, але це займе додатковий час на його перезапис.

Якщо в масиві N елементів, то при зміні його розміру знадобиться виконати $O(N)$ кроків, щоб скопіювати вже присутні елементи на нове місце. Просте резервування місця для додаткових елементів може бути вкрай неефективним.

Припустимо, що в якийсь момент алгоритму знадобиться зберегти 1000 елементів в стеку, хоча більшу частину часу йому потрібно всього кілька. В такому випадку масив стеку буде витрачати невиправдано багато ресурсів пам'яті.

Якщо ж ми знаємо, що в стеці завжди буде зберігатися лише кілька десятків елементів і необхідна висока швидкість при звертанні до нього, то має сенс будувати його саме на основі масиву. В більшості ж випадків стекову структуру слід будувати на основі списків.

Основними операціями над стеком є:

- створення стека;
- включення елемента в стек;
- отримання елемента зі стека;
- видалення стека.

Для векторного стека в пам'яті необхідно побудувати дескриптор і відвести пам'ять для масиву під елементи стека виходячи з максимально можливої їх кількості.

У будь якому випадку при ініціалізації дескриптора в нього обов'язково заносяться покажчики початку і кінця стека (перше і останнє значення індексів масиву), а в покажчику вершини стека встановлюється ознака NULL.

Для стека спискової структури створення стека зводиться до побудови тільки дескриптора, в ньому достатньо наявності лише покажчика на початок списку, який одночасно є і покажчиком вершини стека, так як включення і вибірка елемента здійснюються лише з початку списку. Початково цей покажчик встановлюється в NULL, тим самим позначаючи, що стек порожній.

У разі стека на основі масиву, якщо немає переповнення стека, новий елемент включається на вершину стека при одночасному контролі границі масива, при переповненні – відмова від включення.

У разі стека спискової структури переповнення, як правило, не буває, тим не менш, перевірка необхідна. Включення зводиться до отримання динамічної пам'яті під елемент і занесенню туди даних.

Очевидно, що в покажчик наступного елемента стека в новому елементі заноситься попереднє значення покажчика на вершину стека (початок списку) з дескриптора, а в покажчик вершини стека в дескрипторі – адреса нового елемента.

Вибірка елемента зі стека – операція можлива тільки тоді, коли стек не порожній. Вона означає, що вибирається значення з елемента, що знаходиться на вершині стека, а сам елемент виключається зі стека. Виключення елемента з векторного стека зводиться до зсуву покажчика вершини (індекса масиву) на один елемент до початку стека.

При виключенні елемента зі спискового стека покажчик на вершину стека в дескрипторі замінюється на адресу наступного елемента, а пам'ять колишнього початкового елемента звільняється.

2. Черги

Черга – це змінюваний упорядкований (чи ні) набір елементів.

Додавання елемента в чергу проводиться з одного кінця (хвоста черги, REAR), а вибірка – з іншого кінця (голови черги, FRONT) відповідно до правила "Першим прийшов – першим пішов" (FIFO: First Input – First Output). Така черга є простою чергою без пріоритетів

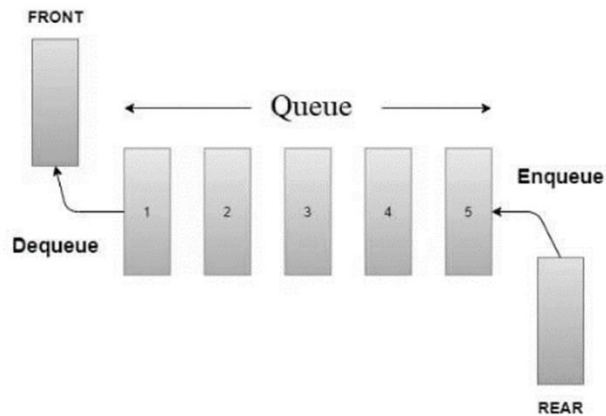


Рисунок 4.4 – Схема простої черги

Пояснимо поняття черг з використанням аналогій, наведених нижче:

- люди на ескалаторі. Люди, які вступили на ескалатор першими будуть першими і при сходженні з нього;
- черга в очікуванні маршрутки. Перша людина, що стоїть в черзі, буде і першою, що зайде в маршрутку;
- черга за квитками біля вікна каси. Перша людина в черзі отримає квиток першою і таким чином, буде першою, яка покине чергу, тощо.

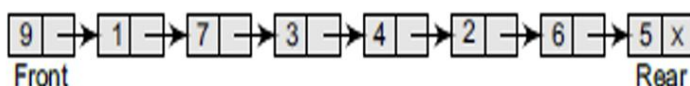
У всіх цих прикладах можеа побачити, що елемент в голові черги обслуговується першочергово. Те ж саме відноситься до структури даних черга. Черги знаходять широке застосування в операційних системах (черги завдань, буфера введення-виведення, буфер введення з клавіатури), в мережах ЕОМ, при моделюванні реальних процесів тощо.

Черги можуть мати векторну або спискову структуру зберігання, в свою чергу векторна структура може займати статичну або динамічну пам'ять. Черга векторної структури через обмеженість елементів має властивість переповнення, коли хвіст черги досягне кінця вектору.

В цьому випадку додавання елементів стає неможливим, навіть якщо в початковій частині вектору будуть вільні місця забраних елементів. Для усунення такого недоліку утворюють кільцеві черги. При досягненні кінця вектору нові елементи додаються на вільні місця з початку вектору.

Тут також можливо переповнення, коли хвіст наздожене голову. Якщо ж голова наздожене хвіст, то черга виявляється порожньою.

Реалізація черги у вигляді масиву ефективна тоді, коли максимальний розмір черги маленький або відомий заздалегідь. Але якщо розмір масиву під чергу не може бути визначено заздалегідь, іншої альтернативи, ніж використання представлення черги у вигляді зв'язного списку, не існує. У такій черзі, кожен елемент складається з двох частин, одна з яких зберігає дані, а інша – адресу наступного елементу.



Показчик START дескриптора черги використовується як FRONT. Крім того, використовується ще один показчик у дескрипторі, званий REAR, який буде зберігати адресу останнього елемента в черзі.

Всі додавання (постановка) нових елементів в чергу будуть виконуватися з хвоста черги, а всі читання (з одночасним видаленням) елементів з черги будуть виконуватися з голови черги. Якщо $FRONT = REAR = NULL$, то це вказує на те, що черга порожня.

Основними операціями над чергою є:

- створення і видалення черги;

- включення в чергу нового елемента;
- вибірка елемента з черги.

При їх виконанні використовуються такі допоміжні операції, як перевірка наявності елементів в черзі і перевірка переповнення черги.

Як і у випадку зі стеком, створення черги зводиться до побудови тільки дескриптора черги. У ньому присутні показчик на голову черги *front*, показчика на хвіст черги *rear* і поточний розмір черги *size*. Показчики *front* і *rear* встановлюється в *NULL*, поле *size* встановлюється в нуль, позначаючи, що черга порожня.

Видалення черги зводиться до звільнення динамічної пам'яті, отриманої в процесі створення і обробки черги. Перед видаленням дескриптора черги слід видалити всі елементи черги по черзі, починаючи з голови черги і до її хвоста. В іншому випадку будуть виникати витoki пам'яті.

Включення в чергу у разі черги спискової структури переповнення, як правило, не буває, тим не менш, перевірка необхідна. Включення в чергу зводиться до отримання динамічної пам'яті під новий елемент, занесенню в нього даних і додавання елемента в хвіст черги.

Очевидно, що в показчик наступного елемента черги в старому хвості заноситься значення показчика на новостворений елемент (новий хвіст черги).

Виключення (читання) елемента з черги – операція можлива тільки тоді, коли черга не порожня. Вона означає, що вибирається значення з елемента, що знаходиться в голові черги, а сам елемент виключається з черги. Показчик на голову черги в дескрипторі замінюється на адресу наступного елемента, а пам'ять під колишню голову черги звільняється.

Черга з пріоритетом є структурою даних черга, в якій кожному елементу присвоюється пріоритет. Пріоритет елемента буде використовуватися для визначення порядку, в якому будуть оброблятися елементи.

Загальні правила обробки елементів в черзі з пріоритетом:

- елемент з більш високим пріоритетом обробляється перед елементом з більш низьким пріоритетом;
- два елементи з однаковим пріоритетом обробляються за принципом перший прийшов - першим обробляється.

Чергу з пріоритетом можна розглядати як модифіковану чергу, в якій для обслуговування вибирається перший елемент з найвищим пріоритетом з головної частини черги. Сам пріоритет елемента може бути встановлений на основі різних факторів.

Черги пріоритетів широко використовуються в операційних системах для виконання процесів з найвищим пріоритетом у першу чергу.

Є два способи реалізації черги з пріоритетами.

Можна використовувати вже *відсортований список* для зберігання елементів. У цьому випадку, коли потрібно витягти елемент з черги, буде вилучатись знову головний елемент, у якого буде максимальний пріоритет після сортування. В цьому випадку при додаванні нового елемента в чергу трудомісткість алгоритму складе $O(N)$, а при вилученні елемента $O(1)$. При іншому підході до організації черги можна використовувати невідсортований зв'язний список елементів. В цьому випадку додавання нового елемента в чергу завжди робиться в хвості черги.

Тоді кожен раз, коли елемент повинен бути взятий з черги на обробку, проводиться пошук в черзі першого елемента з найвищим пріоритетом. При такому підході трудомісткість складе $O(1)$ при додаванні елемента і $O(N)$ при вилученні елемента з черги.

Черги з пріоритетом можуть бути представлені з використанням масивів або зв'язних списків.

Коли черга з пріоритетом реалізується з використанням зв'язного списку структура кожного елемента списку повинна містити наступну інформацію:

- данні елемента черги;

- пріоритет елемента;
- адресу наступного елемента черги.

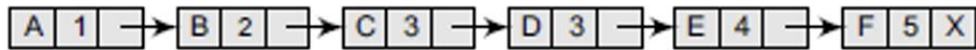


Рисунок 4.5 – Реалізація черги з пріоритетом

Якщо ми використовуємо упорядкований зв'язаний список, то елемент з більш високим пріоритетом буде передувати елементу з більш низьким пріоритетом.

Менше значення поля пріоритету означає більш високий пріоритет. Наприклад, якщо є два елементи A і B, де A має пріоритет 1, а B має пріоритет 2, то A буде оброблений перед B, так як має більш високий пріоритет, ніж B.

З черги, ми не можемо зрозуміти, чи був доданий A перед E, або навпаки, тому що список вже не працює на основі перший прийшов – перший пішов. Тут, елемент з більш високим пріоритетом, передуватиме елементу з більш низьким пріоритетом.

Проте, ми можемо з упевненістю сказати, що C був вставлений в черзі до D, тому що, коли два елементи мають однаковий пріоритет елементи обробляються за принципом перший прийшов – перший пішов.

Створення, видалення черги і читання (вилучення) елемента з черги з пріоритетом виконується так само, як і у випадку простої черги.

Щоб встановити елемент у відсортовану чергу з пріоритетом необхідно в алгоритмі враховувати три випадки:

- черга порожня (елемент додається в початок черги);
- черга не порожня і в черзі немає елементів, пріоритет у яких нижче, ніж у елемента, який додається (елемент додається в хвіст черги);
- черга не порожня і в черзі є елементи з пріоритетом нижче, ніж у того, який додається (елемент стає в середину черги).

В лінійних чергах, обговорюваних досі, включення в чергу може бути виконано тільки в хвіст черги, а виключення елемента – тільки з голови черги.

Подивимося на наступну чергу, для представлення якої будемо використовувати масив, а не зв'язаний список, як раніше

54	9	7	18	14	36	45	21	99	72
0	1	2	3	4	5	6	7	8	9

Видно, що тут FRONT = 0 і REAR = 9. Як видно з малюнка, все повністю заповнено.

Тепер, якщо виникне необхідність поставити в чергу новий елемент, то ця операція стане неможливою: в масиві, яким представлена наша черга, немає більше вільного місця.

Розглянемо тепер ситуацію, в якій зроблені два послідовних читання елементів з черги. Черга після цього стане виглядати так, як показано на наступному рисунку:

		7	18	14	36	45	21	99	72
0	1	2	3	4	5	6	7	8	9

Зараз FRONT = 2 і REAR = 9. Припустимо, що ми тепер хочемо поставити новий елемент в чергу.

Навіть незважаючи на те, що на початку масиву з'явилися два доступних місця, умова переповнення черги, як і раніше залишається, тому що включення в чергу нових елементів ми виконуємо тільки з хвоста.

Це є головним недоліком лінійної векторної черги, реалізованої у вигляді масиву.

Щоб вирішити цю проблему, можна використовувати два підходи.

По перше, перемістити елементи масиву черги вліво, так щоб простір, що звільнився у хвості можна було зайняти і використовувати ефективно. Але такий підхід може зайняти дуже багато часу, особливо, коли розмір черги (масиву) досить великий.

Другий підхід полягає у використанні *кільцевої черги*. У кільцевій черзі, значення першого індексу виникає відразу після проходження останнього індексу.

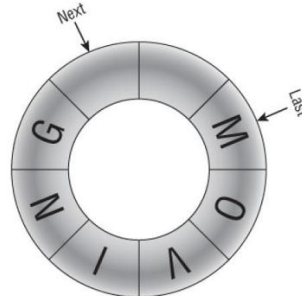


Рисунок 4.6 – Приклад кільцевої черги

Алгоритми реалізації операцій включення і виключення елементів з кільцевої черги будуть складнішими, тому що в них необхідно буде врахувати і запобігати можливій "накладці" голови і хвоста черги при її заповненні, через те, що хвіст і голова черги будуть переміщатися "по колу" в міру включення / виключення елементів з черги. Це є недоліком подібних алгоритмів.

Запитання для самоперевірки

1. Що таке стек?
2. Яка структура стека?
3. Які є способи реалізації стека?
4. Які операції проводяться над стеком?
5. Що таке черга та за яким принципом вона працює?
6. Які є основні операції над чергою?
7. Що таке черга з пріоритетом та її структура?
8. Що таке кільцева черга?

Література

1. Кренивч А.П. Алгоритми і структури даних. Підручник. – К.: ВПЦ «Київський університет», 2021. – 200 с. <https://www.mechmat.univ.kiev.ua/wp-content/uploads/2021/09/pidruchnyk-alhorytmy-i-struktury-danykh.pdf>
4. Barnes G. Data Structures and Algorithms: Annotated Reference with Examples [Електроний ресурс] Режим доступу: <https://mta.ca/~rrosebru/oldcourse/263114/Dsa.pdf>
5. Buki. (n.d.). Python у науці про дані: Як використовувати Python для аналізу даних та машинного навчання. Retrieved September 15, 2025, <https://buki.com.ua/blogs/python-u-nauci-pro-dani-iak-vikoristovuvati-python-dlia-analizu-danix-ta-masinnogo-navcannia>
6. Bethke Uli CSV vs JSON vs XML – The Best Comparison Guide 2026 [Електроний ресурс] Режим доступу: <https://sonra.io/csv-vs-json-vs-xml/>