Основи об’єктно-орієнтованого програмування. Практичні завдання на іспит

Використовуючи принципи ООП, патерни проектування та методи мультипоточного програмування, реалізувати застосунок для демонстрації базових алгоритмів та структур даних.

Реалізувати демонстраційні сутності з певної предметної області (див. Список 1)

Реалізувати наступні базові алгоритми та структури даних:

1. Зв’язний список (див. Список 2). У вузлах списку можуть зберігатись як окремі елементи, так і інші контейнери.
2. Балансоване дерево пошуку (див. Список 3). У вузлах дерева можуть зберігатись як окремі елементи, так і інші контейнери (наприклад, розподіл за полями ключа). Ключем пошуку можуть бути повні сутності, а також окремі поля.
3. Хеш-таблиця на основі Separate Chaining (з можливістю використання довільної структури для вирішення колізій), а також один з варіантів Open Addressing (див. Список 4). В обох випадках передбачити можливість опціонального використання 2-choice hashing.
4. Алгоритми сортування (див. Список 5). Ключем сортування можуть бути повні сутності, а також окремі поля.
5. Використати бібліотечні структури даних (наприклад, std::vector, std::list, std::map, std::unordered_map, чи відповідні аналоги з інших мов програмування), а також бібліотечні алгоритми сортування.

На основі реалізованих та використаних алгоритмів та структур даних реалізувати наступну функціональність:

1. Контейнер «ключ-значення». Реалізувати операції отримання значення за ключем, зміни значення за ключем, додавання значення за ключем, видалення значення за ключем, отримання списку всіх ключів, отримання списку всіх значень, отримання списку всіх пар ключ-значення. Ключами можуть бути примітивні типи, а також реалізовані сутності предметної області чи їх окремі поля. Реалізації на основі хеш-таблиць, балансованих дерев пошуку, впорядкованих списків.
2. Множина. Реалізувати операції перевірки належності елемента множині, додавання елемента, вилучення елемента, операції над множинами – об’єднання, перетин, різниця, симетрична різниця. Елементами множини можуть бути примітивні типи, а також реалізовані сутності предметної області чи їх окремі поля. Реалізації на основі хеш-таблиць, балансованих дерев пошуку, впорядкованих списків.
3. Сортування. Реалізувати операції додавання елемента до контейнера, сортування контейнера за заданим критерієм, отримання заданої кількості елементів з 1) початку, 2) кінця, 3) середини, 4) заданої відносної позиції відсортованого списку

(номер позиції або відсотки). Реалізації на основі списків та балансованих дерев пошуку.

Реалізувати можливість задавати значення вручну, а також генерації заданої кількості випадкових сутностей для перевірки роботи алгоритмів та структур даних.

Реалізувати можливості виміру часу виконання операцій на заданих вхідних даних.

Реалізувати графічний інтерфейс користувача (наприклад, з використанням Qt). Графічний інтерфейс має давати можливість вибору алгоритмів та структур даних, запуску операцій, перегляду результатів та результатів виміру часу роботи.

Реалізувати можливість запуску алгоритмів на окремих потоках, при цьому потік GUI продовжує реагувати на події користувача. Також можна реалізувати паралельні версії алгоритмів, і отримати за це додаткові бали.

Реалізувати unit tests для перевірки роботи реалізованих алгоритмів та структур даних.

Під час реалізації необхідно дотримуватись принципів ООП, зокрема уникати дублювання коду, уникати сильної зв'язаності між різними компонентами, передбачити можливість розширення реалізованої функціональності.

Бажано продемонструвати використання патернів проектування, наприклад:

1. Патерн **Strategy** – варто використати для реалізації алгоритмів, з можливістю заміни реалізації без зміни клієнтського коду
2. Патерн **Template Method** – наприклад, реалізація одних операцій над множинами на основі інших
3. Патерн **Composite** – можливість комбінувати структури даних: у вузлах заданої структури даних зберігаються інші структури даних.
4. Патерн **Decorator** – можливість виміру часу виконання заданого алгоритму чи фрагменту алгоритму.
5. Патерн **Iterator** – варто використати для обходу структур даних.
6. Патерн **Adapter** – варто використати з метою використання бібліотечних реалізацій алгоритмів (зі стандартної бібліотеки або сторонніх бібліотек)
7. Патерн **Abstract Factory** / **Factory Method** – варто використати для побудови стандартних реалізацій певних алгоритмів (залежно від налаштувань)
8. Патерн **Builder** – варто використати для побудови складних структур даних чи алгоритмів
9. Патерн **Singleton** – варто використати для підтримки класів, які мають існувати в одному екземплярі і бути доступними іншим класам (наприклад, класи для побудови алгоритмів)
10. Патерн **Visitor** – варто використати для обробки складних структур даних, наприклад реалізації зберігання, операцій з множинами, …
11. Патерн **Bridge** – варто використати для створення кількох версій інтерфейсу алгоритму чи структури даних, і незалежного розвитку їх реалізацій
12. Патерн **Command** – варто використати для реалізації історії операцій над заданими структурами даних, з можливістю undo/redo
13. Патерн **Memento** – варто використати для збереження та відновлення стану

алгоритмів та структур даних (можливість завершити програму посередині виконання і потім відновитись з тої ж позиції)

14. Патерн **Façade** – варто використати для загального інтерфейсу компонентів та взаємодії компонентів між собою.

Якщо не виходить реалізувати всю описану функціональність повністю, бажано реалізувати хоча б якусь частину функціональності, що демонструє принципи ООП та патерни проектування.

Не ставиться задача реалізувати «з нуля» всі алгоритми та структури даних – можна взяти реалізації з інших завдань (наприклад, лабораторних робіт), чи з інших дисциплін.

Практичну частину іспиту можна виконувати командами до 5 студентів. В цьому разі команда має реалізувати всі сутності та всі алгоритми, які містяться у варіантах студентів-учасників команди. Якщо алгоритми повторюються, можна реалізувати різновиди цих алгоритмів, або інші схожі алгоритми.

Списки варіантів

Список 1. Сутності предметної області

1. Поштова адреса: країна, область/регіон, місто, вулиця, будинок, квартира.
2. Посадові особи: міністерство, організація, підрозділ, співробітник
3. Сервери: компанія, data center, rack, сервер
4. Фрагменти коду: організація, проект, компонент, клас, метод
5. Хмарні сховища: користувач, компанія-провайдер сервісу, каталог, файл, ревізія файлу
6. Книги: видавництво, жанр, рік виходу, автор, назва книги
7. Дата і час: рік, місяць, день, година, хвилина, секунда

Список 2. Зв’язні списки

1. Однозв’язний список
2. Двозв’язний список
3. Однозв’язний циклічний список
4. Двозв’язний циклічний список

Список 3. Балансовані дерева пошуку (реалізувати обидві структури)

1. AVL tree, B tree
2. Red-black tree, B tree
3. Splay tree, B tree
4. AVL tree, B+ tree
5. Red-black tree, B+ tree
6. Splay tree, B+ tree

Список 4. Хеш-таблиці

1. Linear probing
2. Quadratic probing

3. Coalesced hashing
4. Cuckoo hashing
5. Hopscotch hashing

Список 5. Алгоритми сортування (реалізувати всі 4 алгоритми)

1. Selection Sort, Quicksort, Merge Sort, Radix Sort
2. Insertion Sort, Quicksort, Merge Sort, Radix Sort
3. Selection Sort, Quicksort, Heap Sort, Radix Sort
4. Insertion Sort, Quicksort, Heap Sort, Radix Sort
5. Selection Sort, Heap Sort, Merge Sort, Radix Sort
6. Insertion Sort, Heap Sort, Merge Sort, Radix Sort
7. Selection Sort, Quicksort, Merge Sort, Counting Sort
8. Insertion Sort, Quicksort, Merge Sort, Counting Sort
9. Selection Sort, Quicksort, Heap Sort, Counting Sort
10. Insertion Sort, Quicksort, Heap Sort, Counting Sort
11. Selection Sort, Heap Sort, Merge Sort, Counting Sort
12. Insertion Sort, Heap Sort, Merge Sort, Counting Sort
13. Selection Sort, Quicksort, Merge Sort, Bucket Sort
14. Insertion Sort, Quicksort, Merge Sort, Bucket Sort
15. Selection Sort, Quicksort, Heap Sort, Bucket Sort
16. Insertion Sort, Quicksort, Heap Sort, Bucket Sort
17. Selection Sort, Heap Sort, Merge Sort, Bucket Sort
18. Insertion Sort, Heap Sort, Merge Sort, Bucket Sort