Лабораторная работа №2

по курсу
«Информатика (организация и поиск данных)»
(3 семестр)

Варианты заданий

Постановка задачи

Исходной для данной задачи является реализация класса Sequence<T>, полученная в рамках курса за 2-й семестр, а также реализация умных указателей, полученные в ЛР-1 в 3-м семестре. Требуется написать программу на C++ для сравнения различных алгоритмов сортировки. Написать краткое техническое задание (ТЗ). Выполнить реализацию. Написать для нее тесты.

Минимальные требования к программе. В программе должно быть реализовано не менее 2-х различных алгоритмов (см. табл. «Выбор вариантов» ниже). Основные алгоритмы необходимо покрыть тестами. Это касается и реализованного ранее типа Sequence[1]. Программа должна позволять выбрать любой из реализованных алгоритмов сортировки и запустить его на (достаточно произвольных) исходных данных. Алгоритмы сортировки должны быть параметризованы способом сравнения элементов. При этом должна быть возможность как автоматической, так и ручной проверки корректности работы алгоритмов (в т.ч. должна быть возможность просмотра как исходных данных, так и результата – с помощью вывода на экран или/и вывода в файл). Программа должна обладать пользовательским интерфейсом (консольным или графическим). Программа должна позволять проводить проверку работы алгоритмов на длинных последовательностях (10 000 элементов и более); должна быть возможность автоматической генерации последовательностей заданной длины (например, с помощью генератора случайных чисел). Пользовательский интерфейс, в особенности, графический, тестировать не требуется. Программа должна предоставлять функцию измерения времени выполнения алгоритма. Должна быть функция сравнения алгоритмов – по времени выполнения на одних и тех же входных данных[2].

Требования к структуре тестовых данных. В качестве тестовых данных должны ыступать структуры, состояние из по крайней мере 10-15 атрибутов, из которых не менее 5 – должны быть строкового типа, по крайней мере 3 – вещественным числом, а остальные – другими числовыми типами. Типовой пример таких структур – описание Персоны, включающее фамилию, имя и отчество (строковые атрибуты), адрес (строка), год рождения (целочисленный атрибут), рост и вес (вещественные значения), и т.д.. Тестовые данные должны быть заготовлены заранее и храниться в файлах, структура которых разрабатывается студентами.

Требования к процедуре тестирования. Должна быть предусмотрена возможность тестирования в ручном режиме, когда все тестовые данные вводятся с помощью клавиатуры. Это касается всех типов пользовательских интерфейсов, предусмотренных в реализации.

[1] По меньшей мере, должны быть тестами те методы, которые использованы в рамках данной лабораторной работы.

[2] Следует рассматривать три основных случая: массив уже отсортирован в нужном направлении; массив отсортирован в обратном направлении; массив не отсортирован.

Должны быть заранее заготовлены тестовые коллекции, в т.ч. сохраненные в файлах. Должно быть выполнено тестирование на коллекциях объемом 1-50 млн. элементов, а также на файлах объемом «полезной нагрузки» до 1-10 Гб. Должна быть предусмотрена возможность тестирования в автоматизированном режиме, путем прогона набора тестовых коллекций – заготовленных заранее или/и генерируемых на лету.

Требования к реализации сортировок. Для представления данных в памяти используется реализованный ранее абстрактный тип данных – последовательность. Алгоритмы сортировки следует реализовывать инкапсулированными в классах-«сортировщиках», унаследованных от чисто абстрактного класса (интерфейса) ISorter. Сигнатура метода сортировки:

```
Template<typename T>
Sequence<T>* Sort(Sequence<T>* seq, int (*cmp)(T,T));
```

либо:

```
Template<typename T>
Sequence<T>* Sort(Sequence<T>* seq);
```

В последнем случае предполагается, что функция сравнения элементов передается в конструктор класса-«сортировщика». Каждый алгоритм сортировки реализуется в отдельном таком классе, например, алгоритм быстрой сортировки – в классе с названием наподобие `QuickSorter : ISorter<T>`, а пирамидальная – в `PyramidSorter : ISorter<T>`, и т.п.

Однако существенно, что **алгоритм сортировки** так или иначе **должен быть параметризован функцией сравнения сортируемых элементов**. В лабораторной работе должна быть реализована возможность сортировки заготовленных массивов данных по крайней мере по двум разным критериям (по двум разным атрибутам или сочетаниям атрибутов).

При реализации лабораторной работы следует исходить из того, чтобы перечень используемых сортировок не является фиксированным и может быть произвольным образом изменен. Следует принять меры с минимизации усилий по добавлению/удалению алгоритмов из программы.

Требования к представлению результатов сравнения. Замеренное время должно представляться вещественном числом, в миллисекундах. Время выполнения любого фрагмента кода может быть малой величиной, но, очевидно, всегда отлично от нуля. Сравнение следует проводить для серии коллекций объектов данных различного объема, результаты необходимо собрать и представить в табличном и графическом виде.

Т.к. различные алгоритмы сортировки могут кардинально отличаться друг от друга по времени выполнения на некоторых объемах данных, следует предусмотреть механизм, позволяющий: (а) ограничить тестирование более медленных алгоритмах на «слишком больших» для них объемах (желательно, путем оснащения этих реализаций этих алгоритмов соответствующими метаданными) и (б) в табличном и графическом виде отсутствующие данные должны быть отображены адекватно.

Рекомендации. В ряде случаев целесообразно для решения отдельных задач написать отдельные небольшие программы-утилиты с CLI. Для автоматизации многоэтапных процессов удобно написать скрипт, например, на python.

Выбор варианта задания

Каждый студент должен выбрать для реализации 2-3 алгоритма сортировки, исходя из того, чтобы сумма баллов за все выбранные алгоритмы была не менее 14. В таблице ниже приведен список алгоритмов и соответствующие им баллы; количество баллов пропорционально сложности алгоритма.

№	Название алгоритма	Кол-во баллов
1.	Метод пузырька	3
2.	Модификация метода пузырька – шейкерная сортировка	5
3.	Метод простых вставок	5
4.	Сортировка с помощью простого выбора	5
5.	Сортировка подсчетом	7
6.	Метод двоичных вставок	7
7.	Квадратичная сортировка (усов. сортировка выбором)	9
8.	Сортировка с помощью выбора из дерева	7
9.	Сортировка слиянием	7
10.	Пирамидальная сортировка	10
11.	Быстрая сортировка	10
12.	Сортировка Шелла	10
13.	Сортировка Шелла (с выбором смещения)	11
14.	Схема Бэтчера	11

Пример. При выборе метода пузырька, простых вставок и быстрой сортировки сумма баллов: 3+7+10=20.

Опциональные задачи в рамках лабораторной работы (на повышение оценки):

№	Суть задачи	Относительный объем и сложность (max 10)
1.	Реализация плагинной системы для подключения новых алгоритмов сортировки (в т.ч. на «лету»)	9
2.	Реализация «умного» генератора тестовых данных – обеспечивающего желаемые статистические или иные свойства порождаемых массивов данных	8
3.	Поддержка множественных форматов для хранения данных (различных текстовых – csv, xml, json, yaml, и/или бинарных, в т.ч. собственной разработки). Отдельный пункт – работа со строковыми значениями, длина которых заранее не известна.	5
4.	Использование сортировки для ускорения доступа к данным. Подразумевается, что файлы с данными большие – сотни мегабайт и гигабайты, например, сами объекты данных – тоже большие, десятки и сотни байт и более. Вместо того, чтобы при сортировках менять положение объектов, сортируются лишь указатели на положение этих объектов в файлах. Получается нечто вроде простейшего индекса. Реализация требует более тонкой работы с файлами и предполагает поиск требуемого объекта в файле по известному индексу.	5
5.	Визуализация процесса сортировки (на малых объемах, до ~20-30 элементов) с помощью плавной анимации и/или иных средств	6-8

№	Суть задачи	Относительный объем и сложность (max 10)
	визуализации.	
6.	Реализовать подсчет статистик: среднее, медиана, среднеквадратичное, среднеквадратичное отклонение. Наработки могут быть использованы в некоторых вариантах ЛР-3.	3

Критерии оценки

1.	Качество программного кода:	– стиль (в т.ч.: имена, отступы и проч.) (0-2) – структурированность (напр. декомпозиция сложных функций на более простые) (0-2) – качество основных и второстепенных алгоритмов (напр. обработка граничных случаев и некорректных исходных данных и т.п.) (0-2)	0-6 баллов
2.	Качество пользовательского интерфейса:	– предоставляемые им возможности (0-2) – наличие ручного/автоматического ввода исходных данных (0-2) – настройка параметров для автоматического режима – отображение исходных данных и промежуточных и конечных результатов и др. (0-2)	0-6 баллов
3.	Качество тестов	– степень покрытия – читаемость – качество проверки (граничные и некорректные значения, и др.)	0-3 баллов
	Дополнительно		
4.	Владение теорией	знание алгоритмов, области их применимости, умение сравнивать с аналогами, оценить сложность, корректность реализации	0-3 баллов
5.	Оригинальность реализации	оцениваются отличительные особенности конкретной реализации – например, общность структур данных, наличие продвинутых графических средств, средств ввода-вывода, интеграции с внешними системами и др.	0-5 баллов
		Итого	0-15(23)

	баллов

Для получения зачета за выполнения лабораторной работы необходимо соблюдение всех перечисленных условий:

- оценка за п. 1 должна быть не менее 3 баллов
- оценка за п. 2 должна быть не менее 3 баллов
- оценка за п. 3 должна быть больше 0