Формулировки задач по

Конструированию алгоритмов и структур данных

!!!

Каждая задача должна быть загружена на личный git-репозиторий отдельным коммитом, возможно, не одним. Все коммиты должны иметь осмысленные названия и описания того, что в них выполнено. Защита работы возможна на любом лабораторном занятии. Наличие выполненных работ учитывается при выставлении зачёта, а также влияет на итоговую оценку на экзамене.

Если две и более задачи выполнены в один коммит, работа не проверяется. Если все коммиты сделаны в один час, работа не проверяется.

Задачи должны выполняться последовательно, и каждая задача должна быть сдана до начала работы над следующей.

!!!

Задача 18 (отображение на основе дерева поиска).

Определить обобщённый класс MyTreeMap. Класс представляет собой реализацию отображения (карты), которое использует дерево поиска для хранения элементов в строгом порядке.

Необходимы поля:

1) comparator – компаратор для сравнения элементов в отображении;
2) root – корневой узел дерева;
3) size – количество элементов в отображении.

Необходимо реализовать следующую функциональность:

1) конструктор MyTreeMap() для создания пустого отображения, размещающего элементы согласно естественному порядку сортировки;
2) конструктор MyTreeMap(TreeMapComparator comp) для создания пустого отображения, размещающего элементы согласно указанному компаратору;
3) метод clear() для удаления всех пар «ключ-значение» из отображения;
4) метод containsKey(object key) для проверки, содержит ли отображение указанный ключ;

5) метод containsValue(object value) для проверки, содержит ли отображение указанное значение;
6) метод entrySet() для возврата множества (Set) всех пар «ключ-значение» в отображении;
7) метод get(object key) для возврата значения, связанного с указанным ключом, или null, если ключ не найден;
8) метод isEmpty() для проверки, является ли отображение пустым;
9) метод keySet() для возврата множества (Set) всех ключей в отображении;
10) метод put(K key, V value) для добавления пары «ключ-значение» в отображение;
11) метод remove(object key) для удаления пары «ключ-значение» с указанным ключом из отображения;
12) метод size() для возврата количества пар «ключ-значение» в отображении;

13) метод firstKey() для возврата первого ключа отображения;
14) метод lastKey() для возврата последнего ключа отображения;
15) метод headMap(K end) для возврата сортированного отображения, содержащего элементы, ключ которых меньше end;
16) метод subMap(K start, K end) для возврата отображения, содержащего элементы, чей ключ больше или равен start и меньше end;
17) метод tailMap(K start) для возврата сортированного отображения, содержащего элементы, ключ которых больше start;

18) метод lowerEntry(K key) для возврата пары «ключ-значение», где ключ меньше заданного;
19) метод floorEntry(K key) для возврата пары «ключ-значение», где ключ меньше или равен заданному;
20) метод higherEntry(K key) для возврата пары «ключ-значение», где ключ больше заданного;
21) метод ceilingEntry(K key) для возврата пары «ключ-значение», где ключ больше или равен заданному;
22) метод lowerKey(K key) для возврата ключа, который меньше заданного;
23) метод floorKey(K key) для возврата ключа, который меньше или равен заданному;
24) метод higherKey(K key) для возврата ключа, который больше заданного;
25) метод ceilingKey(K key) для возврата ключа, который больше или равен заданному;
26) метод pollFirstEntry() для удаления и возврата первого элемента отображения;

27) метод pollLastEntry() для удаления и возврата последнего элемента отображения;
28) метод firstEntry() для возврата первого элемента отображения без удаления;
29) метод lastEntry() для возврата последнего элемента отображения без удаления.

Предусмотреть обработку возможных ошибок.

Задача 19 (множество на основе красно-чёрного дерева).

Определить обобщённый класс MyTreeSet. Класс представляет собой реализацию множества, которое использует красно-чёрное дерево для хранения элементов в отсортированном порядке по возрастанию.

Необходимы поля:

1) m – объект типа MyTreeMap<E, object> для хранения элементов множества, где ключи представляют элементы множества, а значения – фиктивный объект.

Необходимо реализовать следующую функциональность:

1) Конструктор MyTreeSet() для создания пустого множества, размещающего элементы согласно естественному порядку сортировки.
2) Конструктор MyTreeSet(MyTreeMap<E, object> m) для создания множества, использующего указанный объект MyTreeMap для хранения элементов.
3) Конструктор MyTreeSet(TreeMapComparator comparator) для создания пустого множества, размещающего элементы согласно указанному компаратору.
4) Конструктор MyTreeSet(T[] a) для создания множества, содержащего элементы указанной коллекции.
5) Конструктор MyTreeSet(SortedSet<E> s) для создания множества, содержащего элементы указанного сортированного множества.

6) Метод add(T e) для добавления элемента в конец множества.
7) Метод addAll(T[] a) для добавления элементов из массива.
8) Метод clear() для удаления всех элементов из множества.
9) Метод contains(object o) для проверки, находится ли указанный объект во множестве.
10) Метод containsAll(T[] a) для проверки, содержатся ли указанные объекты во множестве.
11) Метод isEmpty() для проверки, является ли множество пустым.

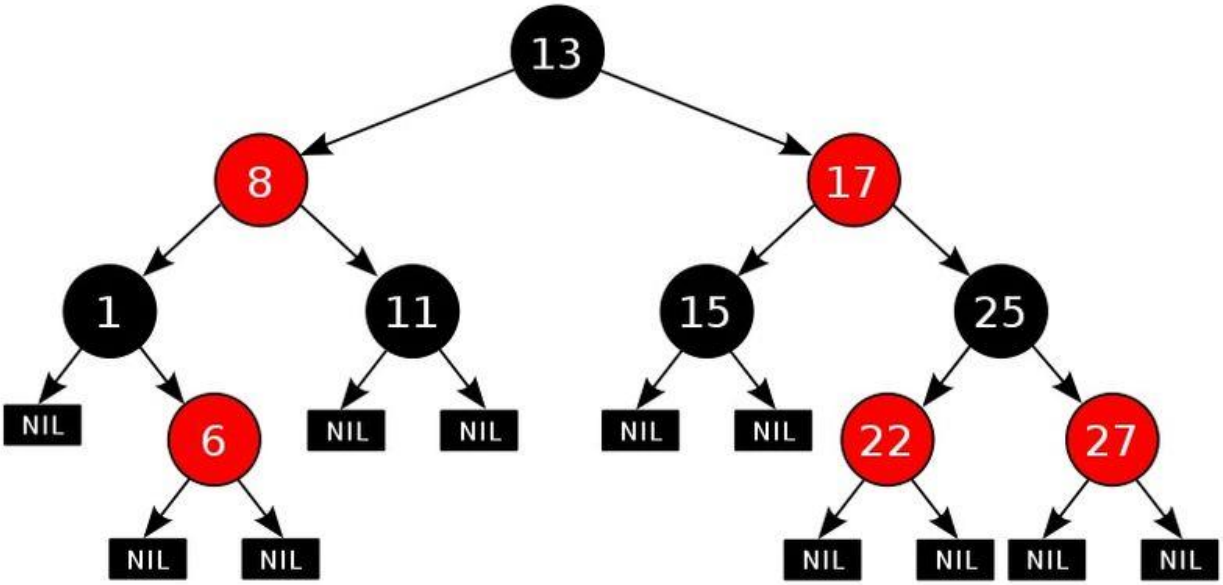
12) Метод remove(object o) для удаления указанного объекта из множества, если он есть там.
13) Метод removeAll(T[] a) для удаления указанных объектов из множества.
14) Метод retainAll(T[] a) для оставления во множестве только указанных объектов.
15) Метод size() для получения размера множества в элементах.
16) Метод toArray() для возвращения массива объектов, содержащего все элементы множества.
17) Метод toArray(T[] a) для возвращения массива объектов, содержащего все элементы множества. Если аргумент a равен null, то создаётся новый массив, в который копируются элементы.

18) Метод first() для возврата первого (наименьшего) элемента множества.
19) Метод last() для возврата последнего (наивысшего) элемента множества.
20) Метод subSet(E fromElement, E toElement) для возврата подмножества элементов из диапазона [fromElement; toElement).
21) Метод headSet(E toElement) для возврата множества элементов, меньших чем указанный элемент.
22) Метод tailSet(E fromElement) для возврата части множества из элементов, больших или равных указанному элементу.

23) Метод ceiling(E obj) для поиска в наборе наименьшего элемента e, для которого истинно e >= obj. Если такой элемент найден, он возвращается. В противном случае возвращается null.
24) Метод floor(E obj) для поиска в наборе наибольшего элемента e, для которого истинно e <= obj. Если такой элемент найден, он возвращается. В противном случае возвращается null.
25) Метод higher(E obj) для поиска в наборе наибольшего элемента e, для которого истинно e > obj. Если такой элемент найден, он возвращается. В противном случае возвращается null.
26) Метод lower(E obj) для поиска в наборе наименьшего элемента e, для которого истинно e < obj. Если такой элемент найден, он возвращается. В противном случае возвращается null.
27) Метод headSet(E upperBound, bool incl) для возврата множества, включающего все элементы вызывающего набора, меньшие upperBound. Результирующий набор поддерживается вызывающим набором.
28) Метод subSet(E lowerBound, bool lowIncl, E upperBound, bool highIncl) для возврата NavigableSet, включающего все элементы вызывающего набора, которые больше lowerBound и меньше

upperBound. Если lowIncl равно true, то элемент, равный lowerBound, включается. Если highIncl равно true, также включается элемент, равный upperBound.

29) Метод tailSet(E fromElement, bool inclusive) для возврата множества, включающего все элементы вызывающего набора, которые больше (или равны, если inclusive равно true) чем fromElement. Результирующий набор поддерживается вызывающим набором.
30) Метод pollLast() для возврата последнего элемента, удаляя его в процессе. Поскольку набор сортирован, это будет элемент с наибольшим значением. Возвращает null в случае пустого набора.
31) Метод pollFirst() для возврата первого элемента, удаляя его в процессе. Поскольку набор сортирован, это будет элемент с наименьшим значением. Возвращает null в случае пустого набора.
32) Метод descendingIterator() для возврата итератора, перемещающегося от большего к меньшему, другими словами, обратного итератора.
33) Метод descendingSet() для возврата множества, представляющего собой обратную версию вызывающего набора. Результирующий набор поддерживается вызывающим набором.



При реализации необходимо учесть следующие правила красно-чёрного дерева:

1) узел может быть либо красным, либо чёрным и имеет двух потомков;
2) корень – как правило чёрный (как правило – потому что, если мы говорим о поддеревьях – это не всегда так);
3) все листья, не содержащие данных – чёрные;

4) оба потомка каждого красного узла – чёрные;
5) любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число чёрных узлов. Простой путь – это тот в котором каждый узел входит ровно по одному разу.

Множество должно обладать следующими преимуществами:

1) Быстрые вставка, удаление и поиск элементов. Скорость операций – $O(\log n)$.

Предусмотреть обработку возможных ошибок.

Задача 20 (графы).

Пусть N – порядковый номер студента в списке его подгруппы. Студенту с номером N необходимо выполнить три задачи из списка, которые определяются по следующим формулам:

1) $\big((N-1)\ \%\ 9\big)+1$;
2) $\big((N-1)\ \%\ 3\big)+10$;
3) $\big((N-1)\ \%\ 6\big)+13$.

Здесь % обозначает операцию взятия остатка от деления.

Список задач:

1) Построение транзитивного замыкания с помощью обхода в глубину.
2) Построение транзитивного замыкания с помощью обхода в ширину.
3) Поиск компонент сильной связности. Алгоритм Мальгранжа.
4) Поиск компонент сильной связности. Алгоритм Косарайю.
5) Поиск компонент сильной связности. Алгоритм Тарьяна.
6) Топологическая сортировка. Алгоритм Тарьяна.
7) Нахождение кратчайших путей от заданной вершины. Алгоритм Дейкстры.
8) Построение минимального остовного леса. Алгоритм Прима.
9) Построение минимального остовного леса. Алгоритм Крускала.
10) Построение максимального потока в транспортной сети. Алгоритм Эдмондса-Карпа.
11) Построение максимального потока в транспортной сети. Алгоритм Диница.
12) Построение максимального потока в транспортной сети. Алгоритм проталкивания предпотока.
13) Построение максимальной клики. Переборный алгоритм.

14) Построение максимальной клики. Эвристический алгоритм «слияния» клик.
15) Построение максимальной клики. Алгоритм Брона-Кербоша.
16) Поиск шарниров в графе.
17) Раскраска графа. Алгоритм Уэлша-Пауэлла.
18) Проверка изоморфности двух графов.

При решении задач следует обратить внимание на эффективное представление графов в памяти компьютера. Там, где это возможно, следует использовать списки смежности для хранения графов. Списки смежности позволяют быстро находить соседей каждой вершины и экономят память по сравнению с матрицей смежности, особенно для разреженных графов.

Задача 21 (отображение на основе хэш-таблицы).

Определить обобщённый класс MyHashMap. Класс представляет собой реализацию отображения (карты) на основе хэш-таблицы, которое может расти по мере необходимости.

Необходимы поля:

1) table – массив обобщённого (универсального) типа Entry[] для хранения пар «ключ-значение»;
2) size – количество пар «ключ-значение» в отображении;
3) loadFactor – коэффициент загрузки, определяющий, когда следует увеличивать размер массива table.

Необходимо реализовать следующую функциональность:

1) конструктор MyHashMap() для создания пустого отображения с начальной ёмкостью 16 и коэффициентом загрузки 0,75;
2) конструктор MyHashMap(int initialCapacity) для создания пустого отображения с указанной начальной ёмкостью и коэффициентом загрузки 0,75;
3) конструктор MyHashMap(int initialCapacity, float loadFactor) для создания пустого отображения с указанной начальной ёмкостью и коэффициентом загрузки;

4) метод clear() для удаления всех пар «ключ-значение» из отображения;
5) метод containsKey(object key) для проверки, содержит ли отображение указанный ключ;
6) метод containsValue(object value) для проверки, содержит ли отображение указанное значение;

7) метод entrySet() для возврата множества (Set) всех пар «ключ-значение» в отображении;
8) метод get(object key) для возврата значения, связанного с указанным ключом, или null, если ключ не найден;
9) метод isEmpty() для проверки, является ли отображение пустым;
10) метод keySet() для возврата множества (Set) всех ключей в отображении;
11) метод put(K key, V value) для добавления пары «ключ-значение» в отображение;
12) метод remove(object key) для удаления пары «ключ-значение» с указанным ключом из отображения;
13) метод size() для возврата количества пар «ключ-значение» в отображении.

Механизм хеширования:

Хеширование – это способ преобразования любой переменной или объекта в уникальный код (хеш-код) с помощью определённой формулы или алгоритма, применяемых к их полям. Хеш-код является уникальным идентификатором содержимого объекта и вычисляется методом hashCode().

1) для одного и того же объекта хеш-код всегда будет одинаковым;
2) если объекты одинаковые (объекты одного класса с одинаковым содержимым полей), то и хеш-коды одинаковые;
3) если хеш-коды равны, то входные объекты не всегда равны (коллизия);
4) если хеш-коды разные, то и объекты гарантированно разные.

Итог:

1) если хеш-коды разные, то и входные объекты гарантированно разные;
2) если хеш-коды равны, то входные объекты не всегда равны (коллизия).

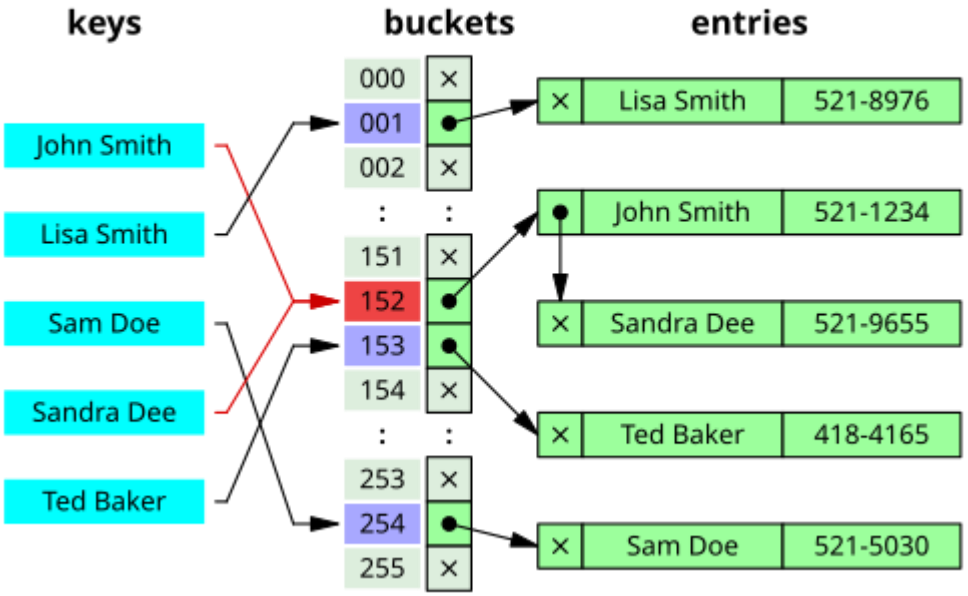
Хеш-таблица:

Хеш-таблица – это обобщение обычного массива, где ключом может быть любой объект, для которого можно вычислить хеш-код. Ячейки массива называются корзинами и содержат связанный список объектов, реализованный по принципу LinkedList. Список содержит пары «ключ-значение» для Map, а для Set ключом является сам объект с нулевым значением.

Процесс добавления нового элемента в HashMap с помощью метода put(key, value):

1) Вычисляется значение hashCode у ключа с помощью одноимённого метода.

2) Определяется бакет (ячейка массива), в которую будет добавлен новый элемент. Номер бакета определяется по остатку от деления хэшкода на количество ячеек.
3) Если бакет пустой, то элемент просто добавляется. Если бакет не пустой, то в нём находится LinkedList.
4) Если бакет не пустой, мы идём по этому списку и сравниваем ключ добавляемого элемента и ключ элемента в списке по хэшкодам.
5) Если хэшкоды неравны, то идём к следующему элементу.
6) Если хэшкоды равны, то далее сравниваем ключи по методу equals().
7) Если ключи равны по equals(), то перезаписываем значение (value) по этому ключу.
8) Если ключи не равны по equals(), то переходим к следующему элементу.
9) Если мы не нашли ключ в списке, то добавляем новый элемент в конец списка.



Предусмотреть обработку возможных ошибок.

Задача 22 (считывание из файла в отображение (HTML-теги)).

Из файла input.txt считывать строки (без пробелов). Из каждой строки с помощью регулярных выражений извлечь теги (Пример: <html>, </H1>, <PrIvet>). Тег начинается символом <, заканчивается символом >, после < может иметь символ /. Остальные символы – цифры и буквы, первый символ – обязательно буква. С помощью отображения подсчитать количество вхождений каждого из тегов (теги сравниваются без учёта регистра и наличия/отсутствия символа ‘/’).

Для хранения и обработки данных использовать собственную реализацию отображения MyHashMap. Использование встроенного отображения приведёт к незачёту задачи.

Задача 23 (считывание из файла в отображение (переменные)).

В файле находятся определения переменных в формате:

название_типа имя_переменной = значение;

Определение может быть более чем на одной строке в исходной файле. Считать, что файл состоит только из определений в таком формате. Названия типа и имя переменной – идентификаторы, а значение – целое беззнаковое число.

Считать все данные из файла. Для выделения отдельного определения использовать регулярные выражения (а затем и отдельных его частей). Полученные определения поместить в хеш-таблицу (если они корректны), в следующем виде: ключ – это название переменной, значение – тип переменной (сделать enum) и значение (в виде строки). Определение считается корректным, если тип один из следующий: int, float, double. О всех некорректных определениях необходимо сообщить.

Кроме того, необходимо сообщить о всех переопределениях переменных, то есть переменных с одинаковыми именами. При этом необходимо оставлять в таблице первую определённую переменную.

Результат вывести в файл в виде строк:

название типа => имя_переменной(значение)

Для хранения и обработки данных использовать собственную реализацию отображения MyHashMap. Использование встроенного отображения приведёт к незачёту задачи.

Задача 24 (сравнение отображения на основе хеш-таблицы и отображения на основе дерева поиска).

Сравнить эффективность отображения на основе хеш-таблицы (MyHashMap) и отображения, которое использует дерево поиска (MyTreeMap) для различных операций на разных размерах данных структур.

Описание:

1) Сравнить эффективность отображения на основе хеш-таблицы и отображения на основе дерева поиска для операций получения значения

по ключу (get), присваивания значения по ключу (put), удаления элемента по ключу (remove).

2) Для операции присваивания значения по ключу (put) создать пустые отображения на основе хеш-таблицы и дерева поиска и выполнить 10^5, 10^6, 10^7, 10^8 операций добавления элементов с уникальными ключами.
3) Для операций получения значения по ключу (get) и удаления элемента по ключу (remove) создать отображения на основе хеш-таблицы и дерева поиска размером 10^5, 10^6, 10^7, 10^8 элементов.
4) Для каждого размера отображений на основе хеш-таблицы и дерева поиска и каждой операции провести 20 запусков и вычислить среднее время выполнения.
5) Представить результаты сравнения в виде графиков, показывающих, какая из данных структур (отображение на основе хеш-таблицы или отображение на основе дерева поиска) работает эффективнее для каждой операции на разных размерах отображений.
6) Проанализировать полученные результаты и сделать выводы об эффективности использования отображения на основе хеш-таблицы и отображения на основе дерева поиска для различных операций на разных размерах данных структур. Объяснить, почему одна структура данных может быть более эффективной для определённых операций, чем другая, и как это связано с реализацией каждой структуры данных.

Задача 25 (множество на основе отображения на основе дерева поиска).

Определить обобщённый класс MyHashSet. Класс представляет собой реализацию множества, которое может расти по мере необходимости.

Необходимы поля:

1) map – объект типа MyHashSet<E, object> для хранения элементов множества, где ключи представляют элементы множества, а значения – фиктивный объект.

Необходимо реализовать следующую функциональность:

1) Конструктор MyHashSet() для создания пустого множества с начальной ёмкостью 16 и коэффициентом загрузки 0,75.
2) Конструктор MyHashSet(T[] a) для создания множества, содержащего элементы указанной коллекции.
3) Конструктор MyHashSet(int initialCapacity, float loadFactor) для создания пустого множества с указанной начальной ёмкостью и коэффициентом загрузки.

4) Конструктор MyHashSet(int initialCapacity) для создания пустого множества с указанной начальной ёмкостью и коэффициентом загрузки 0,75.

5) Метод add(T e) для добавления элемента в конец множества.
6) Метод addAll(T[] a) для добавления элементов из массива.
7) Метод clear() для удаления всех элементов из множества.
8) Метод contains(object o) для проверки, находится ли указанный объект во множестве.
9) Метод containsAll(T[] a) для проверки, содержатся ли указанные объекты во множестве.
10) Метод isEmpty() для проверки, является ли множество пустым.
11) Метод remove(object o) для удаления указанного объекта из множества, если он есть там.
12) Метод removeAll(T[] a) для удаления указанных объектов из множества.
13) Метод retainAll(T[] a) для оставления во множестве только указанных объектов.
14) Метод size() для получения размера множества в элементах.
15) Метод toArray() для возвращения массива объектов, содержащего все элементы множества.
16) Метод toArray(T[] a) для возвращения массива объектов, содержащего все элементы множества. Если аргумент a равен null, то создаётся новый массив, в который копируются элементы.

Предусмотреть обработку возможных ошибок.

Задача 26 (считывание из файла во множество (сравнение строк)).

Из файла input.txt считать строки с пробелами и поместить их в множество. Строки сравнивать по длине слов входящих в них (если самое короткое слово одной строки короче, чем самое короткое слово второй строки, то строка меньше; если их длины равны, то тоже самое для вторых по длине слов и т. д., если в одной строке закончились слова, а во второй – нет, то первая меньше). Словом считать последовательность символов без пробелов. Для хранения слов строки использовать список (его нужно отсортировать).

Для хранения и обработки данных использовать собственную реализацию множества MyHashSet. Использование встроенного множества приведёт к незачёту задачи.

Задача 27 (считывание из файла во множество (уникальные слова)).

В файле находятся строки, содержащие слова (и возможно что-нибудь другое). Словом считать последовательности латинских букв. С помощью хеш-множества найти все уникальные слова (с точностью до регистра, т.е. “abc” и “AbC” – одинаковые слова).

Для хранения и обработки данных использовать собственную реализацию множества MyHashSet. Использование встроенного множества приведёт к незачёту задачи.

Задача 28 (итераторы).

Определить интерфейс MyIterator, который определяет методы hasNext(), next(), remove().

В классах MyPriorityQueue, MyArrayDeque, MyHashSet, MyTreeSet реализовать класс MyItr, который реализует интерфейс MyIterator.

Необходимы поля:

1) cursor – указатель на текущий элемент коллекции.

Необходимо реализовать следующие методы:

1) hasNext() – возвращает true, если в коллекции имеется следующий элемент, иначе возвращает false;
2) next() – возвращает следующий элемент коллекции;
3) remove() – удаляет текущий элемент из коллекции.

В классах MyPriorityQueue, MyArrayDeque, MyHashSet, MyTreeSet необходимо реализовать следующую функциональность:

1) метод iterator() для получения итератора для прохода по всем элементам с возможностью вставки или замены.

Определить интерфейс MyIterator, который определяет методы hasNext(), next(), hasPrevious(), previous(), nextIndex(), previousIndex(), remove(), set(T element), add(T element).

В классах MyArrayList, MyVector, MyLinkedList реализовать класс MyItr, который реализует интерфейс MyIterator.

Необходимы поля:

1) cursor – указатель на текущий элемент коллекции.

Необходимо реализовать следующие методы:

1) hasNext() – возвращает true, если в коллекции имеется следующий элемент, иначе возвращает false;
2) next() – возвращает следующий элемент коллекции;
3) hasPrevious() – возвращает true, если в коллекции имеется предыдущий элемент, иначе возвращает false;
4) previous() – возвращает предыдущий элемент коллекции;
5) nextIndex() – возвращает индекс следующего элемента;
6) previousIndex() – возвращает индекс предыдущего элемента;
7) remove() – удаляет текущий элемент из коллекции;
8) set(T element) – заменяет текущий элемент коллекции на указанный элемент;
9) add(T element) – вставляет указанный элемент в коллекцию перед элементом, который будет возвращён следующим вызовом next().

В классах MyArrayList, MyVector, MyLinkedList необходимо реализовать следующую функциональность:

1) метод listIterator() для получения итератора для прохода по всем элементам с возможностью вставки или замены;
2) метод listIterator(int index) для получения итератора с указанной позиции.

Возможные ошибки объединить в иерархию и обеспечить их обработку с помощью полиморфизма.

Задача 29 (интерфейсы).

Определить иерархию интерфейсов и классов, которые реализуют различные структуры данных:

1) определить интерфейс MyCollection, который определяет методы add(T e), addAll(T[] a), clear(), contains(object o), containsAll(T[] a), isEmpty(), remove(object o), removeAll(T[] a), retainAll(T[] a), size(), toArray(), toArray(T[] a);
2) определить интерфейс MyList, который расширяет MyCollection и определяет методы add(int index, T e), addAll(int index, T[] a), get(int index), indexOf(object o), lastIndexOf(object o), listIterator(), listIterator(int index), remove(int index), set(int index, T e), subList(int fromIndex, int toIndex);
3) классы MyArrayList, MyVector, MyLinkedList – реализации интерфейса MyList;

4) определить интерфейс MyQueue, который расширяет MyCollection и определяет методы element(), offer(T obj), peek(), poll();
5) класс MyPriorityQueue – реализация интерфейса MyQueue;
6) определить интерфейс MyDeque, который расширяет MyCollection и определяет методы addFirst(T obj), addLast(T obj), getFirst(), getLast(), offerFirst(T obj), offerLast(T obj), pop(), push(T obj), peekFirst(), peekLast(), pollFirst(), pollLast(), removeLast(), removeFirst(), removeLastOccurrence(object obj), removeFirstOccurrence(object obj);
7) класс MyArrayDeque – реализация интерфейсов MyList и MyDeque;
8) определить интерфейс MySet, который расширяет MyCollection и определяет методы first(), last(), subSet(E fromElement, E toElement), headSet(E toElement), tailSet(E fromElement);
9) класс MyHashSet – реализация интерфейса MySet;
10) определить интерфейс MySortedSet, который расширяет MySet и определяет методы first(), last(), subSet(E fromElement, E toElement), headSet(E toElement), tailSet(E fromElement);
11) определить интерфейс MyNavigableSet, который расширяет MySortedSet и определяет методы lowerEntry(K key), floorEntry(K key), higherEntry(K key), ceilingEntry(K key), lowerKey(K key), floorKey(K key), higherKey(K key), ceilingKey(K key), pollFirstEntry(), pollLastEntry(), firstEntry(), lastEntry();
12) класс MyTreeSet – реализация интерфейса MyNavigableSet;
13) определить интерфейс MyMap, который определяет методы clear(), containsKey(object key), containsValue(object value), entrySet(), get(object key), isEmpty(), keySet(), put(K key, V value), putAll(Map<K, V> m), remove(object key), size(), values();
14) класс MyHashMap – реализация интерфейса MyMap;
15) определить интерфейс MySortedMap, который расширяет MyMap и определяет методы firstKey(), lastKey(), headMap(K end), subMap(K start, K end), tailMap(K start);
16) определить интерфейс MyNavigableMap, который расширяет MySortedMap и определяет методы lowerEntry(K key), floorEntry(K key), higherEntry(K key), ceilingEntry(K key), lowerKey(K key), floorKey(K key), higherKey(K key), ceilingKey(K key), pollFirstEntry(), pollLastEntry(), firstEntry(), lastEntry();
17) класс MyTreeMap – реализация интерфейса MyNavigableMap.

Вместо параметра T[] a необходимо использовать параметр MyCollection<T> c:

1) в интерфейсе MyCollection в методах containsAll(T[] a), removeAll(T[] a), retainAll(T[] a);
2) в интерфейсе MyList в методе addAll(int index, T[] a);

3) в классе MyArrayList в конструкторе MyArrayList(T[] a);
4) в классе MyVector в конструкторе MyVector(T[] a);
5) в классе MyPriorityQueue в конструкторе MyPriorityQueue(T[] a);
6) в классе MyArrayDeque в конструкторе MyArrayDeque(T[] a);
7) в классе MyLinkedList в конструкторе MyLinkedList(T[] a);
8) в классе MyHashSet в конструкторе MyHashSet(T[] a);
9) в классе MyTreeSet в конструкторе MyTreeSet(T[] a).

Необходимо реализовать следующую функциональность:

1) в классе MyHashMap метод putAll(MyMap<K, V> m) для добавления всех пар «ключ-значение» из указанного отображения в текущее отображение;
2) в классе MyHashMap метод values() для возврата коллекции (MyCollection) всех значений в отображении;
3) в классе MyTreeMap конструктор MyTreeMap(MyMap<K, V> m) для создания отображения, содержащего элементы указанной карты;
4) в классе MyTreeMap конструктор MyTreeMap(MySortedMap<K, V> sm) для создания отображения, содержащего элементы указанной сортированной карты;
5) в классе MyTreeMap метод putAll(MyMap<K, V> m) для добавления всех пар «ключ-значение» из указанного отображения в текущее отображение;
6) в классе MyTreeMap метод values() для возврата коллекции (MyCollection) всех значений в отображении.

Возможные ошибки объединить в иерархию и обеспечить их обработку с помощью полиморфизма.

Разработать UML-диаграмму классов.

Задача 30 (строка).

Определить класс MyString. Класс представляет собой реализацию строки.

Поля класса:

1) value – массив для хранения символов строки;
2) length – длина строки (количество символов).

Конструкторы:

1) MyString() – создаёт пустую строку;
2) MyString(char[] value) – создаёт строку из массива символов;
3) MyString(MyString original) – создаёт копию переданной строки.

Основные методы:

1) length() – возвращает длину строки;
2) charAt(int index) – возвращает символ по указанному индексу;
3) substring(int beginIndex, int endIndex) – возвращает подстроку;
4) concat(MyString str) – объединяет строки;
5) equals(MyString str) – проверяет равенство строк;
6) equalsIgnoreCase(MyString str) – проверяет равенство строк без учёта регистра;
7) toLowerCase() – преобразует строку в нижний регистр;
8) toUpperCase() – преобразует строку в верхний регистр;
9) trim() – удаляет пробелы в начале и конце строки;
10) replace(char oldChar, char newChar) – заменяет символы в строке;
11) contains(MyString substr) – проверяет наличие подстроки;
12) indexOf(MyString substr) – возвращает индекс первого вхождения подстроки.

Дополнительные методы:

1) split(char delimiter) – разбивает строку по указанному разделителю;
2) startsWith(MyString prefix) – проверяет, начинается ли строка с указанного префикса;
3) endsWith(MyString suffix) – проверяет, заканчивается ли строка указанным суффиксом;
4) reverse() – возвращает строку в обратном порядке.

Статические методы:

1) valueOf(int i) – преобразует число в строку;
2) valueOf(double d) – преобразует число с плавающей точкой в строку;
3) valueOf(boolean b) – преобразует логическое значение в строку.

Требования:

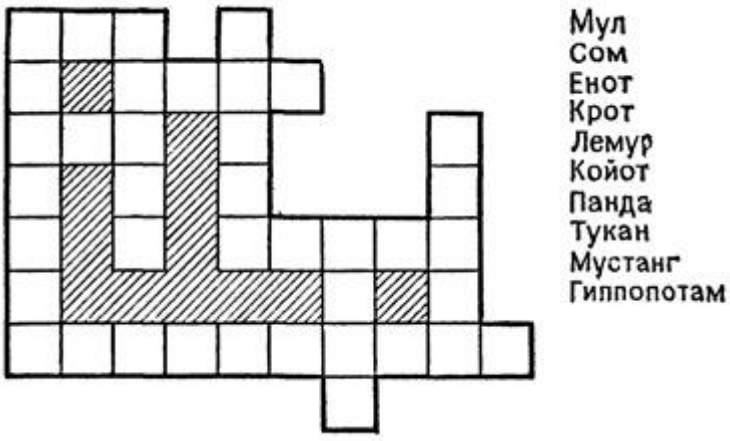
1) для хранения символов строки использовать массив char[];
2) предусмотреть обработку ошибок, таких как выход за границы массива или передача null в качестве аргумента.

Задача 31* (крисс-кросс).

Необходимо разработать программу, которая по заданному списку слов строит корректную и связную схему головоломки типа «крисс-кросс».

Крисс-кросс – это головоломка, похожая на кроссворд, но без определений. Игроку предоставляется пустая сетка и список слов, которые нужно вписать в

сетку так, чтобы они пересекались по правилам кроссворда: в местах пересечения у слов должны совпадать буквы. Все слова из списка должны быть использованы, и каждое из них может быть использовано только один раз. Длина слов служит подсказкой для их размещения.



Ввод данных:

1) Программа должна читать список слов, разбитых на группы по длине и упорядоченных по алфавиту внутри каждой группы.
2) Список слов может содержать слова разной длины.

Построение схемы:

1) Сгенерировать связную схему крисс-кросса, в которую будут вписаны все слова из списка.
2) В местах пересечения слов должны совпадать соответствующие буквы.
3) Схема должна быть связной, то есть все слова должны быть соединены между собой через пересечения.
4) В схеме не должно быть повторяющихся слов.

Вывод результатов:

1) Представить заполненную схему как подтверждение правильности решения.
2) Обеспечить красивый графический вывод схемы для наглядности (например, используя текстовую графику или графический интерфейс).

Обработка особых случаев:

1) Если для заданного списка слов невозможно построить связную схему крисс-кросса, программа должна сообщить об этом.
2) В случаях, когда возникает неоднозначность (например, существует несколько вариантов размещения слов или повторяются слова), программа должна определить и указать на эту проблему.

Оптимизация схемы:

1) Постараться увеличить «связанность» схемы. Это означает, что слова должны быть максимально переплетены между собой.
2) Связанность можно измерять следующими способами:
    a. Среднее число пересечений на слово.
    b. Отношение площади, занимаемой словами, к площади наименьшего прямоугольника, охватывающего всю схему.
    c. Минимальное число пересечений на слово.
3) Разработать эвристики для выбора очередного слова и его размещения, чтобы увеличить связанность схемы.

Алгоритмическая часть:

1) Использовать метод перебора с возвратами (backtracking) для построения схемы.
2) Разработать эффективные структуры данных для хранения и быстрого доступа к информации о словах и их возможных позициях на сетке.
3) Контролировать однозначность решения, проверяя, что никакие два слова равной длины нельзя поменять местами без нарушения условий головоломки.

Требования к оформлению:

1) Разместить исходный код программы в репозитории на Git с понятной структурой и комментариями.
2) Подготовить отчёт с подробным описанием:
    a. Постановки задачи и её анализа.
    b. Выбранных алгоритмов и структур данных.
    c. Особенностей реализации программы.
    d. Проведённых тестов и полученных результатов.
    e. Выводов по проделанной работе.
3) Отчёт должен быть грамотно оформлен, содержать оглавление, введение, основную часть и заключение. Рекомендуется использовать форматирование для улучшения читаемости (разделы, списки, изображения схемы и т.д.).

Для лучшего понимания задачи и возможного подхода к её решению рекомендуется ознакомиться со статьёй https://habr.com/ru/articles/166471/.

Решение данной задачи представляет интерес не только с практической точки зрения, но и с теоретической, поскольку полноценное алгоритмическое решение, максимизирующее связанность схемы, является сложной и нетривиальной задачей.