Оценочные материалы для промежуточной аттестации по дисциплине

Алгоритмы и структуры данных

Квалификация выпускника	бакалавр
Направление подготовки	01.03.02 «ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА»
Направленность (профиль)	«Технологии программирования и анализ данных»
Форма обучения	очная
Кафедра-разработчик	Прикладной математики
Выпускающая кафедра	Прикладной математики

Типовые задания для контрольной работы (3 семестр).

Задание: адаптируя стандартные структуры данных и алгоритмы написать программу, позволяющую заданную прикладную задачу, оценить ее эффективность и составить отчет по результатам ее применения.

Вариант 1. Тимофей ищет место, чтобы построить себе дом. Улица, на которой он хочет жить, имеет длину *n*, то есть состоит из *n* одинаковых идущих подряд участков. Каждый участок либо пустой, либо на нём уже построен дом.

Общительный Тимофей не хочет жить далеко от других людей на этой улице. Поэтому ему важно для каждого участка знать расстояние до ближайшего пустого участка. Если участок пустой, эта величина будет равна нулю — расстояние до самого себя.

Помогите Тимофею посчитать искомые расстояния. Для этого у вас есть карта улицы. Дома в городе Тимофея нумеровались в том порядке, в котором строились, поэтому их номера на карте никак не упорядочены. Пустые участки обозначены нулями.

Вариант 2. Тимофей решил организовать соревнование по спортивному программированию, чтобы найти талантливых стажёров. Задачи подобраны, участники зарегистрированы, тесты написаны. Осталось придумать, как в конце соревнования будет определяться победитель.

Каждый участник имеет уникальный логин. Когда соревнование закончится, к нему будут привязаны два показателя: количество решённых задач P_i и размер штрафа F_i. Штраф начисляется за неудачные попытки и время, затраченное на задачу.

Тимофей решил сортировать таблицу результатов следующим образом: при сравнении двух участников выше будет идти тот, у которого решено больше задач. При равенстве числа решённых задач первым идёт участник с меньшим штрафом. Если же и штрафы совпадают, то первым будет тот, у которого логин идёт раньше в алфавитном (лексикографическом) порядке.

Тимофей заказал толстовки для победителей и накануне поехал за ними в магазин. В своё отсутствие он поручил вам реализовать алгоритм быстрой сортировки (*англ.* quick sort) для таблицы результатов. Так как Тимофей любит спортивное программирование и не любит зря расходовать оперативную память, то ваша реализация сортировки не может потреблять $O(n)$ дополнительной памяти для промежуточных данных (такая модификация быстрой сортировки называется "in-place").

Как работает in-place quick sort

Как и в случае обычной быстрой сортировки, которая использует дополнительную память, необходимо выбрать опорный элемент (*англ.* pivot), а затем переупорядочить массив. Сделаем так, чтобы сначала шли элементы, не превосходящие опорного, а затем —— большие опорного.

Затем сортировка вызывается рекурсивно для двух полученных частей. Именно на этапе разделения элементов на группы в обычном алгоритме используется дополнительная память. Теперь разберёмся, как реализовать этот шаг *in-place*.

Пусть мы как-то выбрали опорный элемент. Заведём два указателя *left* и *right*, которые изначально будут указывать на левый и правый концы отрезка соответственно.

Затем будем двигать левый указатель вправо до тех пор, пока он указывает на элемент, меньший опорного. Аналогично двигаем правый указатель влево, пока он стоит на элементе, превосходящем опорный. В итоге окажется, что что левее от *left* все элементы точно принадлежат первой группе, а правее от *right* — второй. Элементы, на которых стоят указатели, нарушают порядок. Поменяем их местами (в большинстве языков программирования используется функция *swap()*) и продвинем указатели на следующие элементы. Будем повторять это действие до тех пор, пока *left* и *right* не столкнутся.

Вариант 3. Гоша реализовал структуру данных Дек, максимальный размер которого определяется заданным числом.
Методы push_back(x), push_front(x), pop_back(), pop_front() работали корректно. Но, если в деке было много элементов, программа работала очень долго. Дело в том, что не все операции выполнялись за *O(1)*. Помогите Гоше! Напишите эффективную реализацию.
Внимание: при реализации используйте кольцевой буфер.

Вариант 4. Тимофей, как хороший руководитель, хранит информацию о зарплатах своих сотрудников в базе данных и постоянно её обновляет. Он поручил вам написать реализацию хеш-таблицы, чтобы хранить в ней базу данных с зарплатами сотрудников.
Хеш-таблица должна поддерживать следующие операции:

- put key value —– добавление пары ключ-значение. Если заданный ключ уже есть в таблице, то соответствующее ему значение обновляется.
- get key —– получение значения по ключу. Если ключа нет в таблице, то вывести «None». Иначе вывести найденное значение.
- delete key —– удаление ключа из таблицы. Если такого ключа нет, то вывести «None», иначе вывести хранимое по данному ключу значение и удалить ключ.

В таблице хранятся уникальные ключи.

Требования к реализации:

- Нельзя использовать имеющиеся в языках программирования реализации хеш-таблиц (std::unordered_map в C++, dict в Python, HashMap в Java, и т. д.)
- Разрешать коллизии следует с помощью метода цепочек или с помощью открытой адресации.
- Все операции должны выполняться за *O(1)* в среднем.
- Поддерживать рехеширование и масштабирование хеш-таблицы не требуется.
- Ключи и значения, id сотрудников и их зарплата, —– целые числа. Поддерживать произвольные хешируемые типы не требуется.

Типовые задания для курсового проекта (4 семестр)

Задание: адаптируя стандартные структуры данных и алгоритмы написать программу, позволяющую заданную прикладную задачу, оценить ее эффективность и составить отчет по результатам ее применения.

Вариант 1. Дано бинарное дерево поиска, в котором хранятся ключи. Ключи — уникальные целые числа. Найдите вершину с заданным ключом и удалите её из дерева так, чтобы дерево осталось корректным бинарным деревом поиска. Если ключа в дереве нет, то изменять дерево не надо.

На вход вашей функции подаётся корень дерева и ключ, который надо удалить. Функция должна вернуть корень изменённого дерева. Сложность удаления узла должна составлять O(h), где h —— высота дерева. Создавать новые вершины (вдруг очень захочется) нельзя.

Вариант 2. Тимофей решил соединить все компьютеры в своей компании в единую сеть. Для этого он придумал построить минимальное остовное дерево, чтобы эффективнее использовать ресурсы.

Но от начальства пришла новость о том, что выделенный на сеть бюджет оказался очень большим и его срочно надо израсходовать. Поэтому Тимофея теперь интересуют не минимальные, а максимальные остовные деревья.

Он поручил вам найти вес такого максимального остовного дерева в неориентированном графе, который задаёт схему офиса.

Вариант 3. На Алгосах устроили турнир по настольному теннису. Гоша выиграл n партий, получив при этом некоторое количество очков за каждую из них.

Гоше стало интересно, можно ли разбить все заработанные им во время турнира очки на две части так, чтобы сумма в них была одинаковой.

Вариант 4. Вам даны строки в запакованном виде. Определим запакованную строку (ЗС) рекурсивно. Строка, состоящая только из строчных букв английского алфавита является ЗС. Если A и B —— корректные ЗС, то и AB является ЗС. Если A —— ЗС, а n — однозначное натуральное число, то $n[A]$ тоже ЗС. При этом запись $n[A]$ означает, что при распаковке строка A записывается подряд n раз. Найдите наибольший общий префикс распакованных строк и выведите его (в распакованном виде).

Иными словами, пусть сложение —— это конкатенация двух строк, а умножение строки на число — повтор строки соответствующее число раз. Пусть функция f умеет принимать ЗС и распаковывать её. Если ЗС D имеет вид $D=AB$, где A и B тоже ЗС, то $f(D) = f(A) + f(B)$. Если $D=n[A]$, то $f(D) = f(A) \times n$.

Этап: проведение промежуточной аттестации по дисциплине

Семестр 3 (Экзамен)

Задание для показателя оценивания дескриптора «Знает»	Вид задания
Сформулируйте развернутые ответы на следующие теоретические вопросы (при необходимости проиллюстрировать и привести примеры): 1. Линейный и бинарный поиск. 2. Оценка сложности алгоритма. 3. Тестирование алгоритмов. 4. Эффективный ввод-вывод. 5. Оперативная память и представление данных. 6. Массивы постоянного размера. 7. Динамические массивы. 8. Списки. 9. Стек. 10. Очередь. 11. Дек. 12. Стек вызовов, рекурсия, переполнение. 13. Примеры задач на рекурсию. 14. Рекурсивный и базовый случаи. 15. Бинарный поиск и рекурсия. 16. Сортировки вставками. 17. Сортировка по ключу. 18. Сравнение элементов. 19. Сортировка слиянием. 20. Быстрая сортировка. 21. Сортировка подсчетом. 22. Ассоциативный массив. 23. Хеш-таблица и хеш-функция. 24. Выбор размера хеш-таблицы и вычисление номера корзины. 25. Свойства хеш-функции. 26. Коллизии. 27. Метод цепочек. 28. Метод открытой адресации. 29. Идеальное хеширование.	- теоретический

30. Построение хеш-функций для строк. 31. Поисковый индекс.	

Задание для показателя оценивания дескриптора «Умеет», «Владеет»	Вид задания
Адаптируя стандартные структуры данных и алгоритмы написать программу, позволяющую заданную прикладную задачу, оценить ее эффективность и составить отчет по результатам ее применения. Варианты прикладных задач: 1. Задание связано с обратной польской нотацией. Она используется для парсинга арифметических выражений. Еще её иногда называют постфиксной нотацией. В постфиксной нотации операнды расположены перед знаками операций. Пример 1: *3 4 +* означает *3 + 4* и равно *7* Пример 2: *12 5 /* Так как деление целочисленное, то в результате получим 2. Пример 3: *10 2 4 * -* означает *10 - 2 * 4* и равно *2* Разберём последний пример подробнее: Знак * стоит сразу после чисел *2* и *4*, значит к ним нужно применить операцию, которую этот знак обозначает, то есть перемножить эти два числа. В результате получим 8. После этого выражение приобретёт вид: *10 8 -* Операцию «минус» нужно применить к двум идущим перед ней числам, то есть *10* и *8*. В итоге получаем *2*. Рассмотрим алгоритм более подробно. Для его реализации будем использовать стек. Для вычисления значения выражения, записанного в обратной польской нотации, нужно считывать выражение слева направо и придерживаться следующих шагов: 1. Обработка входного символа: • Если на вход подан операнд, он помещается на вершину стека. • Если на вход подан знак операции, то эта операция выполняется над требуемым количеством значений, взятых из стека в порядке добавления. Результат выполненной операции помещается на вершину стека. 2. Если входной набор символов обработан не полностью, перейти к шагу 1.	- практический

3. После полной обработки входного набора символов результат вычисления выражения находится в вершине стека. Если в стеке осталось несколько чисел, то надо вывести только верхний элемент.

Замечание про отрицательные числа и деление**:** в этой задаче под делением понимается математическое целочисленное деление. Это значит, что округление всегда происходит вниз. А именно: если $a / b = c$, то $b \cdot c$ — это наибольшее число, которое не превосходит a и одновременно делится без остатка на b.

Например, *-1 / 3 = -1*. Будьте осторожны: в C++, Java и Go, например, деление чисел работает иначе.

В текущей задаче гарантируется, что деления на отрицательное число нет.

2. Игра «Тренажёр для скоростной печати» представляет собой поле из клавиш 4x4. В нём на каждом раунде появляется конфигурация цифр и точек. На клавише написана либо точка, либо цифра от 1 до 9.

В момент времени t игрок должен одновременно нажать на все клавиши, на которых написана цифра t. Гоша и Тимофей могут нажать в один момент времени на k клавиш каждый. Если в момент времени t нажаты все нужные клавиши, то игроки получают 1 балл.

Найдите число баллов, которое смогут заработать Гоша и Тимофей, если будут нажимать на клавиши вдвоём.

3. Алла ошиблась при копировании из одной структуры данных в другую. Она хранила массив чисел в кольцевом буфере. Массив был отсортирован по возрастанию, и в нём можно было найти элемент за логарифмическое время. Алла скопировала данные из кольцевого буфера в обычный массив, но сдвинула данные исходной отсортированной последовательности. Теперь массив не является отсортированным. Тем не менее, нужно обеспечить возможность находить в нем элемент за $O(\log n)$. Можно предполагать, что в массиве только уникальные элементы.

4. В этой задаче можно пользоваться хеш-таблицами из стандартных библиотек.

Тимофей пишет свою поисковую систему.

Имеется n документов, каждый из которых представляет собой текст из слов. По этим документам требуется построить поисковый индекс. На вход системе будут подаваться запросы. Запрос —– некоторый набор слов. По запросу надо вывести 5 самых релевантных документов.

Релевантность документа оценивается следующим образом: для каждого уникального слова из запроса берётся число его вхождений в документ, полученные числа для всех слов из запроса суммируются. Итоговая сумма и является релевантностью

документа. Чем больше сумма, тем больше документ подходит под запрос. Сортировка документов на выдаче производится по убыванию релевантности. Если релевантности документов совпадают —— то по возрастанию их порядковых номеров в базе (то есть во входных данных). Подумайте над случаями, когда запросы состоят из слов, встречающихся в малом количестве документов. Что если одно слово много раз встречается в одном документе?	

Семестр 4 (Экзамен)

Задание для показателя оценивания дескриптора «Знает»	Вид задания
Сформулируйте развернутые ответы на следующие теоретические вопросы (при необходимости проиллюстрировать и привести примеры): 1. Деревья. 2. Двоичные деревья поиска. 3. Обход дерева. 4. Вставка элемента. 5. Удаление элемента. 6. Сбалансированные деревья поиска. 7. Приоритетная очередь. 8. Вставка и удаление. 9. Куча. 10. Пирамидальная сортировка. 11. Представление графов в памяти. 12. DFS-обход в глубину. 13. Поиск цикла и времени входа-выхода. 14. Топологическая сортировка. 15. Связность неориентированного графа. 16. BFS-обход в ширину. 17. Алгоритм Дейкстры. 18. Минимальное островное дерево. 19. Жадные алгоритмы. 20. Примеры задач. 21. Динамическое программирование.	- теоретический

22. Алгоритм решения. 23. Двумерная динамика. 24. Динамическое решение прикладных задач. 25. Поиск наибольшей общей последовательности, наибольшей общей возрастающей последовательности. 26. Простейшие операции со строками. 27. Сравнение строк. 28. Подстроки, префиксы, суффиксы. 29. Поиск шаблона в строке. 30. Префикс-функция. 31. Вычисление префикс-функции. 32. Эффективный поиск шаблона в тексте. 33. Префиксное дерево.	

Задание для показателя оценивания дескриптора «Умеет», «Владеет»	Вид задания
Адаптируя стандартные структуры данных и алгоритмы написать программу, позволяющую заданную прикладную задачу, оценить ее эффективность и составить отчет по результатам ее применения. Варианты прикладных задач: 1. В данной задаче необходимо реализовать сортировку кучей. При этом кучу необходимо реализовать самостоятельно, использовать имеющиеся в языке реализации нельзя. Тимофей решил организовать соревнование по спортивному программированию, чтобы найти талантливых стажёров. Задачи подобраны, участники зарегистрированы, тесты написаны. Осталось придумать, как в конце соревнования будет определяться победитель. Каждый участник имеет уникальный логин. Когда соревнование закончится, к нему будут привязаны два показателя: количество решённых задач P_i и размер штрафа F_i. Штраф начисляется за неудачные попытки и время, затраченное на задачу. Тимофей решил сортировать таблицу результатов следующим образом: при сравнении двух участников выше будет идти тот, у которого решено больше задач. При равенстве числа решённых задач первым идёт участник с меньшим штрафом. Если же и штрафы совпадают, то первым будет тот, у которого логин идёт раньше в алфавитном (лексикографическом) порядке. Тимофей заказал толстовки для победителей и накануне поехал за ними в магазин. В своё отсутствие он поручил вам реализовать алгоритм сортировки кучей (*англ.* Heapsort) для таблицы результатов.	- практический

2. В стране X есть n городов, которым присвоены номера от 1 до n. Столица страны имеет номер n. Между городами проложены железные дороги.

Однако дороги могут быть двух типов по ширине полотна. Любой поезд может ездить только по одному типу полотна. Условно один тип дорог помечают как R, а другой как B. То есть если маршрут от одного города до другого имеет как дороги типа R, так и дороги типа B, то ни один поезд не сможет по этому маршруту проехать. От одного города до другого можно проехать только по маршруту, состоящему исключительно из дорог типа R или только из дорог типа B.

Но это ещё не всё. По дорогам страны X можно двигаться только от города с меньшим номером к городу с большим номером. Это объясняет большой приток жителей в столицу, у которой номер n.

Карта железных дорог называется оптимальной, если не существует пары городов A и B такой, что от A до B можно добраться как по дорогам типа R, так и по дорогам типа B. Иными словами, для любой пары городов верно, что от города с меньшим номером до города с бОльшим номером можно добраться по дорогам только какого-то одного типа или же что маршрут построить вообще нельзя. Выясните, является ли данная вам карта оптимальной.

3. Расстоянием Левенштейна между двумя строками s и t называется количество атомарных изменений, с помощью которых можно одну строку превратить в другую. Под атомарными изменениями подразумеваются: удаление одного символа, вставка одного символа, замена одного символа на другой. Найдите расстояние Левенштейна для предложенной пары строк. Выведите единственное число — расстояние между строками.

4.Вася готовится к экзамену по алгоритмам и на всякий случай пишет шпаргалки. Чтобы уместить на них как можно больше информации, он не разделяет слова пробелами. В итоге получается одна очень длинная строка. Чтобы на самом экзамене из-за нервов не запутаться в прочитанном, он просит вас написать программу, которая по этой длинной строке и набору допустимых слов определит, можно ли разбить текст на отдельные слова из набора.

Более формально: дан текст T и набор строк $s_1, \ldots, s_n$. Надо определить, представим ли T как $s_{k1}s_{k2}\ldots s_{kr}$, где где k_i — индексы строк. Индексы могут повторяться. Строка s_i может встречаться в разбиении текста T произвольное число раз. Можно использовать не все строки для разбиения. Строки могут идти в любом порядке