Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный университет имени П.М. Машерова»
Кафедра прикладной математики и механики

О.Г. Казанцева

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Методические рекомендации по выполнению лабораторных работ

Витебск
ВГУ имени П.М. Машерова
2016

УДК 004.42(076.5)
ББК 32.972.34я73
К14

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 2 от 24.12.2015 г.

Авторы: старший преподаватель кафедры прикладной математики и механики ВГУ имени П.М. Машерова **О.Г. Казанцева**

Р е ц е н з е н т :
доцент кафедры автоматизации технологических процессов и производства УО «ВГТУ», кандидат технических наук, доцент *В.Е. Казаков*

Казанцева, О.Г.

К14 Алгоритмы и структуры данных : методические рекомендации по выполнению лабораторных работ / О.Г. Казанцева. – Витебск : ВГУ имени П.М. Машерова, 2016. – 52 с.

В методических рекомендациях представлены теоретические сведения, практические задания, указания по выполнению лабораторных работ, задания для самостоятельной работы, примерные варианты контрольных работ, примерный перечень теоретических и практических вопросов, выносимых на зачет. Эти материалы помогут студенту освоить изучаемые темы и выполнить лабораторные работы по различным темам курса «Алгоритмы и структуры данных». В материалах содержатся описание структур данных, базовых операций над рассматриваемыми структурами данных, а также алгоритмы решения типовых задач по каждой теме.

Предназначается для студентов специальностей «Прикладная математика», «Прикладная информатика» (дисциплина «Алгоритмы и структуры данных»), «Программное обеспечение информационных технологий» («Основы алгоритмизации и программирования»).

УДК 004.42(076.5)
ББК 32.972.34я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данного издания – представить методические рекомендации по выполнению лабораторных работ в курсе «Алгоритмы и структуры данных». Учебное издание содержит краткую теоретическую информацию о линейных и нелинейных структурах данных, базовых операциях работы с ними, алгоритмах сортировки и поиска, подходах, используемых при разработке эффективных алгоритмов.

Особое внимание в данных методических материалах уделяется анализу алгоритмов и выбору способа их реализации. Эффективное решение любой задачи по программированию зависит не только от удачного алгоритма решения, но также и от правильного выбора структуры данных, позволяющей максимально эффективно реализовывать основные операции алгоритма: добавление и удаление элементов, поиск, последовательность обработки элементов. Построение корректной математической модели, обоснованный выбор структуры данных, вычисление оценок сложности алгоритма, использование принципов «динамическое программирование», «разделяй и властвуй» способствуют разработке эффективных алгоритмов.

Использование основ объектно-ориентированного программирования позволяет обеспечить высокий уровень реализации изучаемых алгоритмов, поэтому студентам рекомендуется при выполнении лабораторных работ вести разработку программ на любом объектно-ориентированном языке программирования.

При рассмотрении тем фрагменты операций и алгоритмов приводятся на псевдокоде, чтобы свести до минимума различие при использовании разных языков программирования. Во всех темах приведены задания для самостоятельной работы студентов, которые могут также использоваться для оценки уровня усвоения материала.

Контроль выполнения заданий на самостоятельную работу проводится в форме защиты отчёта по выполнению лабораторной работы.

Материал соответствует отдельным темам рабочих программ курсов: «Алгоритмы и структуры данных» (специальности «Прикладная математика», «Прикладная информатика»), «Основы алгоритмизации и программирования» (специальность «Программное обеспечение информационных технологий»).

ТРЕБОВАНИЯ, ПРЕДЪЯВЛЯЕМЫЕ К ПРИЛОЖЕНИЯМ, РАЗРАБАТЫВАЕМЫМ В РАМКАХ КУРСА

1. Задания, в которых требуется разработать приложение или реализовать алгоритм, необходимо разрабатывать на объектно-ориентированном языке программирования, следуя принципам объектно-ориентированного проектирования.
2. Обосновывайте выбор структур данных и оценивайте сложность алгоритма. Разрабатывайте наиболее эффективный алгоритм.
3. Исходные данные могут быть произвольных типов, поэтому, при разработке приложений используйте параметризованные классы.
4. Используйте систему контроля версий. Приветствуется совместная работа над проектом командой из двух человек.
5. Для тестирования приложения, подготовьте набор разнообразных входных данных, покрывающих все множество возможных входных данных. Для генерации случайных правдоподобных входных данных разработайте утилиту (класс + методы).
6. При работе с программой пользователь должен видеть условие задачи, требования к входным данным, информативные подсказки, в случае некорректного ввода.
7. Разработанная программа должна обеспечивать возможность работы в двух режимах: а) ручной ввод данных – выполнять алгоритм для одного набора входных данных (в этом случае ввод данных и вывод результата осуществляется в полях формы или консоль), б) ввод данных из файла, вывод результата – в файл.

Основные требования к оформлению кода:

1. Программа, написанная на языке java должна соответствовать «java code conventions». Программы на других языках должны следовать этим требования, насколько это возможно.
2. Каждая команда должна быть представлена на отдельной строке. Любое объявление свойства или переменной также должно быть выполнено на отдельной строке.
3. Структура программы должна быть ярко выражена с помощью использования необходимого количества отступов в строке (от 2х до 4х), для отражения структурной вложенности языковых конструкций.
4. Код программ необходимо сопровождать комментариями, в количестве, достаточном для быстрого понимания алгоритма. Счетчики в циклах традиционно называют `i`, `j`, `k`, `l`, `m`, `n`.
5. Вид доступа к члену класса всегда должен быть явно указан.
6. Смысл названий классов, свойств, методов, переменных должны быть понятен при чтении (самодокументирующийся код). В названиях следовать стилю «Camel», т.е первая буква строчная, а остальные первые буквы слов заглавные. Например, `setName`, `bubbleSort`.

ЛАБОРАТОРНАЯ РАБОТА № 1
АНАЛИЗ АЛГОРИТМОВ

1.1. Трудоемкость алгоритма

Под ***алгоритмом*** понимается единый метод решения определенного класса однотипных задач, обладающий свойством дискретности, массовости, определенности, результативности и оперирующий конструктивными объектами.

Под ***размерностью задачи***, которую в дальнейшем будем обозначать через l, понимают то количество информации, которое достаточно для формального описания задачи.

Мы будем полагать, что размерность задачи – это минимальное количество бит, которого достаточно для описания входных данных задачи. ***Зам.*** *Далее предполагаем, что размерность машинного слова достаточна для представления любого числа, и размерность задачи будет ограничена количеством исходных данных в ее формальном описании.*

Трудоемкость алгоритма – это функция от размерности задачи, которая оценивает сверху время, требуемое для решения задачи.

Трудоемкость алгоритма (сложность алгоритма) измеряется двумя параметрами: $T(l)$ (временная сложность), $S(l)$ (пространственная, емкостная сложность, или требования к памяти).

Учитывая сделанное ранее предположение о том, что для представления любого числа нам достаточно одного машинного слова, получаем, что емкостная сложность алгоритма (размерность памяти) для задания n элементов не превосходит n.

Если в качестве модели вычислений взять неветвяшуюся программу и предположить, что алгоритм – это последовательность арифметических операций и все арифметические операции (аддитивные: +, inc, dec; мультипликативные: *, /) эквивалентны, т. е. затрачивают на свое выполнение одну единицу времени (хотя на самом деле известно, что мультипликативные операции работают дольше), то трудоемкость (временная сложность) алгоритма определяется как функция с количеством операций, требующимся для решения задачи, выраженным через ее размерность.

Однако, при оценке трудоемкости алгоритма точное значение количества операций не имеет для нас существенного значения, более важна скорость роста этого числа при возрастании объема входных данных, т.е. нас интересуют асимптотические оценки.

Алгоритм называется ***полиномиальным***, если его трудоемкость $T(l) = O(p(l))$, где $p(1)$ – некоторый полином или полиномиально ограниченная функция.

Алгоритм называется ***экспоненциальным***, если его трудоемкость $T(l) = \Omega(\exp(l))$, где $\exp(l)$ – некоторая экспоненциальная функция.

1.2. Рекуррентные уравнения

Для некоторой задачи под *подзадачей* мы будем понимать ту же задачу, но меньшим числом параметров, *или* задачу с тем же числом параметров, но при этом хотя бы один из параметров имеет меньшее значение.

Соотношения, связывающие одни и те же функции, но с различными аргументами, называются ***рекуррентными уравнениями***.

Рекуррентное уравнение называется ***правильным*** если значения аргументов у любой из функций правой части соотношения меньше значения аргументов любой из функций левой части соотношения; если аргументов несколько, то достаточно уменьшение одного из них.

Правильное рекуррентное уравнение называется ***полным***, если оно определено для всех допустимых значений аргументов.

$T(n) = T(n-1) + T(n-2)$, $T(1)= 1$, $T(2)=1$.

Методы решений рекуррентных уравнений:

1. Оценка решения рекуррентного уравнения. Метод подстановок.
2. Метод итераций.
3. Метод рекурсивных деревьев.

1.3. Самостоятельная работа «Асимптотический анализ»

Для выполнения работы рекомендуется ознакомится с главами 1, 2, 3, 4 (п.4.3 – 4.6) [1]. ***Зам***. 1) *В книге [1, стр. 33] для обозначения $\log_2 n$ используется обозначение $\lg n$*; 2) *микросекунда – одна миллионная доля секунды (0,000001, или 10^{-6})).*

Вариант 1

1. Предположим, на одной и той же машине проводится сравнительный анализ реализаций двух алгоритмов сортировки, работающих по методу вставок и по методу слияния. Для сортировки n элементов методом вставок необходимо $8n^2$ шагов, а для сортировки методом слияния – $64n \lg n$ шагов. При каком значении n время сортировки методом вставок превысит время сортировки методом слияния?
2. Расположите следующие функции по порядку в соответствии с нотацией большого О (обоснуйте это расположение). Сгруппируйте, например, с помощью подчеркивания, функции, которые являются Θ оценкой друг другу:
   а) $200n^3 \lg n$, б) $2n^{100}$, в) $n \lg n$, г) $4^{\lg n}$, д) n^3 , е) $(\lg n)^{\lg n}$, ж) $n^{\lg \lg n}$.
3. Оцените скорость роста функций $f(n) = \{n, \lg n, n \lg n, n^2, 2^n, n!\}$ при росте $n = 2^0, 2^4, 2^8, 2^{12}, 2^{16}, 2^{32}$. Результат представьте в виде таблицы.

4. Пусть $f(n)$ и $g(n)$ – асимптотически неотрицательные функции. Докажите с помощью базового определения Θ-обозначений, что $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.
5. Пусть $f(n)$ и $g(n)$ – асимптотически положительные функции. Докажите или опровергните справедливость каждого из приведенных ниже утверждений:
   а. Из $f(n) = O(g(n))$ следует, что $g(n) = O(f(n))$.
   б. $f(n) + g(n) = \Theta(\min(f(n), g(n))$.
   в. Из $f(n) = O(g(n))$ следует $\lg(f(n)) = O(\lg(g(n))$, где при достаточно больших n выполнены неравенства $\lg(g(n)) \geq 1$ и $f(n) \geq 1$.
   г. Из $f(n) = O(g(n))$ следует $2^{f(n)} = O(2^{g(n)})$.
6. Выразите функцию $f(n) = n^3/1000 - 100\,n^2 - 100\,n + 3$ в Θ обозначениях.
7. Решите следующие рекуррентные уравнения: $T(n) = T(n-1) + n-1$, $n > 2$, $T(1) = 3$.
8. Определите верхнюю и нижнюю асимптотические границы функции $T(n)$ для каждого из представленных ниже рекуррентных соотношений. Считаем, что $T(n)$ – константа при $n \leq 2$. Обоснуйте свой ответ.
   а) $T(n) = T(9n/10) + n$, б) $T(n) = 6\,T(n/2) + n^2$, в) $T(n) = 16\,T(n/4) + n^2$.

Вариант 2

1. При каком минимальном значении n алгоритм, время работы которого определяется формулой $100n^2$, работает быстрее, чем алгоритм, время работы которого выражается как 2^n, если оба алгоритма выполняются на одной и той же машине?
2. Расположите следующие функции по порядку в соответствии с нотацией большого О (обоснуйте это расположение). Сгруппируйте, например, с помощью подчеркивания, функции, которые являются Θ оценкой друг другу:
   а) $10\,n \lg n$, б) 2^{100}, в) $200^{\lg n}$, г) $n\,2^n$, д) $\lg(n!)$, е) $n!$, ж) $n^{\lg \lg n}$.
3. Для каждой функции $f(n) = \{\lg n, n^{1/2}, n, n \lg n, n^2, 2^n, n!\}$ и времени t (1 секунда, 1 час, 1 неделя, 1 год, 1 столетие) определите наибольший размер n задачи, которая может быть решена за время t, при условии, что для ее решения алгоритму необходимо $f(n)$ микросекунд. Результат представьте в виде таблицы.
4. Покажите, что для любых действительных констант a и b, где $a, b > 0$, справедливо соотношение $(n+a)^b = \Theta(n^b)$.
5. Пусть $f(n)$ и $g(n)$ – асимптотически положительные функции. Докажите или опровергните справедливость каждого из приведенных ниже утверждений:
   а. $f(n) = O(f(n)^2)$.
   б. Из $f(n) = O(g(n))$ следует, что $g(n) = \Omega(f(n))$.
   в. $f(n) = \Theta(f(n/2))$.

г. Из $f(n) + o(f(n)) = \Theta(f(n))$.

6. Выразите функцию $f(n)=3n^2/1000+10\,n\,\lg n+200\,n+3$ в Θ обозначениях.
7. С помощью метода рекурсивных деревьев, докажите, что решение рекуррентного соотношения $T(n) = T(n/3) + T(2n/3) + c\,n$ ведет себя как $\Omega(n \lg n)$, где c – константа.
8. Определите верхнюю и нижнюю асимптотические границы функции $T(n)$ для каждого из представленных ниже рекуррентных соотношений. Считаем, что $T(n)$ – константа при достаточно малых n. Обоснуйте свой ответ.
а) $T(n) = 2\,T(n/2)+ n^3$, б) $T(n) = 7\,T(2n/5)+ n^2$, в) $T(n) = 2\,T(n/4)+ n^{1/2}$.

1.4. Самостоятельная работа «Разработка алгоритма»

При решении задач, предложите несколько алгоритмов решения задачи, для каждого из них определите *размерность задачи, число арифметических операций, трудоемкость алгоритма*. Ответьте на вопрос – является ли алгоритм полиномиальным или экспоненциальным. Выберите для реализации наиболее эффективный алгоритм.

Вариант 1

1. Разработать алгоритм вычисления факториала: $n!$.
2. Разработать приложение для сложения/вычитания двух целых чисел, длиной не более чем n цифр каждое, в системе счисления k (k от 2 до 10), хранящихся в n-элементных массивах A и B ($0 \le n \le 20\,000$). Сумму/разность этих двух чисел необходимо занести в двоичной форме в $(n + 1)$-элементный массив C. ***Зам***. *При выводе результата незначащие цифры не выводить*.

Вариант 2

1. Разработать алгоритм определения простоты числа n.
2. Разработать приложение для поиска отсутствующего целого числа в массиве $A[1..n]$, содержащем все целые числа от 0 до n за исключением одного ($0 \le n \le 20\,000$). При этом действует ограничение: нет доступа к целому числу из массива A посредством одной операции. Элементы массива A представлены в двоичной системе счисления, и единственная операция, с помощью которой можно осуществлять к ним доступ, это «извлечение j-го бита элемента $A[i]$», которая выполняется в течение фиксированного времени. **Зам**. Алгоритм поиска отсутствующего числа должен иметь трудоемкость $O(n)$. Можно использовать вспомогательный массив $B[1..n]$, предназначенный для записи имеющихся чисел из массива A.

Контрольные вопросы:

1. Дайте определение понятия «алгоритм», определите его свойства. Раскройте смысл этих свойств с помощью примеров.
2. Чем отличается современная трактовка понятия алгоритма от значения этого слова в прошлом? Чем можно объяснить историческое изменение значения этого понятия?
3. Какие способы описания алгоритмов существуют.
4. Что означает понятие «правильный алгоритм»?
5. Перечислите этапы разработки программ.
6. Приведите примеры семантических, синтаксических, логических ошибок в программе.
7. Дайте определение понятию «количество информации». Определите какого количества информации достаточно для установления номера выпавшего значения кубика.
8. Дайте определение понятию «размерность задачи».
9. Дайте определение понятию «сложность алгоритмы». С какой целью проводится анализ сложности алгоритма и зачем применяется система сравнительных оценок алгоритмов?
10. Определите понятия асимптотических оценок $f(n)=O(g(n))$, $f(n)=\Omega(g(n))$, $f(n)=\Theta(g(n))$. С какой целью проводится асимптотический анализ функций трудоёмкости алгоритмов.
11. Сформулируйте свойства транзитивности, рефлексивности и симметричности асимптотических оценок.
12. Дайте определение понятий «экспоненциальный алгоритм», «полиномиальный алгоритм». Приведите примеры.
13. Объясните понятие «равнодоступная адресная машина». Приведите примеры ее использования.
14. Определите трудоемкость последовательной конструкции «ветвление» и «цикл».
15. Определите трудоемкость конструкции «цикл» со вложенным циклом.
16. Определите трудоемкость конструкции «цикл» с k вложенными циклами.
17. Дайте определение понятию «рекуррентное уравнение». Приведите примеры рекуррентных уравнений.
18. Какое рекуррентное уравнение называется правильным? Приведите примеры правильных рекуррентных уравнений.
19. Опишите «метод итераций» для решения рекуррентных уравнений.
20. Опишите метод оценки решения рекуррентного уравнения: «метод подстановок».
21. Опишите «метод рекурсивных деревьев» для решения рекуррентных уравнений.
22. Сформулируйте и докажите основную теорему о решении рекуррентного уравнения.

ЛАБОРАТОРНАЯ РАБОТА № 2
СОРТИРОВКИ

2.1. Разработка эффективных алгоритмов. Подход «разделяй и властвуй»

Многие полезные алгоритмы имеют рекурсивную структуру: для решения данной задачи они рекурсивно вызывают сами себя один или несколько раз, чтобы решить вспомогательную задачу, имеющую непосредственное отношение к поставленной задаче. Такие алгоритмы зачастую разрабатываются с помощью *метода декомпозиции*, или *разбиения*: сложная задача разбивается на несколько более простых, которые подобны исходной задаче, но имеют меньший объем; далее эти вспомогательные задачи решаются рекурсивным методом, после чего полученные решения комбинируются с целью получить решение исходной задачи.

Парадигма, лежащая в основе метода декомпозиции «разделяй и властвуй», на каждом уровне рекурсии включает в себя три этапа.

- *Разделение* задачи на несколько подзадач (непересекающихся).
- *Покорение* – рекурсивное решение этих подзадач. Когда объем подзадачи достаточно мал, выделенные подзадачи решаются непосредственно.
- *Комбинирование* решения исходной задачи из решений вспомогательных задач.

При разбиении задачи на подзадачи полезен ***принцип балансировки***, который предполагает, что задача разбивается на подзадачи приблизительно равных размерностей, т.е. идет поддержание равновесия. Обычно такая стратегия приводит к разделению исходной задачи пополам и обработке каждой из ее частей тем же способом до тех пор, пока части не станут настолько малыми, что их можно будет обрабатывать непосредственно. Часто такой процесс приводит к логарифмическому множителю в формуле, описывающей трудоемкость алгоритма.

2.2. Задача сортировки

Частичным порядком на множестве *S* называется такое бинарное отношение *R*, что для любых *a, b* и *c* из S

1) *aRa* (*R* рефлексивно),
2) *aRb* и *bRc* => *aRc* (*R* транзитивно),
3) *aRb* и *bRa* => *a=b* (*R* антисимметрично).

Линейным, или ***полным порядком*** на множестве *S* называется такой частичный порядок *R* на *S*, что для любых двух элементов *a*, *b* выполняется либо *aRb*, либо *bRa* *(другими словами, элементы a, b сравнимы)*

Задача: Пусть дана последовательность из n элементов $\{a_1, a_2, \ldots, a_n\}$ выбранных из множества, на котором задан линейный порядок. элемент a_i назовем записью, линейный порядок будем обозначать $\leq$

Каждая запись a_i имеет ключ k_i, который управляет процессом сортировки, помимо ключа, запись может иметь некоторую дополнительную информацию, которая не влияет на процесс сортировки, но всегда присутствует в этой записи.

Требуется найти такую перестановку $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ этих n записей, после которой ключи расположились бы в неубывающем порядке: $k_{\pi(1)} \leq k_{\pi(2)} \leq \ldots \leq k_{\pi(n)}$.

Алгоритм сортировки называется ***устойчивым***, если в процессе сортировки относительное расположение элементов одинаковыми ключами не изменяется (предполагается, что элементы уже были отсортированы по некоторому вторичному ключу).

$\pi(i) < \pi(j)$, если $k_{\pi(i)} \leq k_{\pi(j)}$ и $i < j$.

2.3. Самостоятельная работа «Внутренняя сортировка»

1. Реализовать 2 алгоритма сортировки, в соответствии с вариантами, определяемыми преподавателем.
2. Подготовить отчет, в котором описать реализованные алгоритмы, использованные структуры данных, оценить сложность алгоритмов, описать худшие и лучшие случаи входных данных, определить обеспечивают ли реализованные вами алгоритмы устойчивую сортировку.
3. Провести экспериментальное сравнение производительности работы алгоритмов:
   - оценить число операций сравнения и число операций обмена (перемещений) элементов, (***Зам***. *Например, используйте глобальные счетчики, которые увеличиваются в соответствующих методах*.)
   - время работы (***Зам***. *Реализовать метод time( )*).
4. Сравнение методов сортировки провести для n = 10, 100, 1 000, 10 000 и следующем порядке входных элементов:
   - элементы уже упорядочены;
   - элементы упорядочены в обратном порядке;
   - расстановка элементов случайна.

Зам. Учитывайте, что сравнение методов сортировки проводится для одинаковых входных данных.

5. Результаты экспериментов оформить на основе нескольких запусков программы в виде двух сводных таблиц по образцу:

Таблица 1 – Название метода сортировки

n	Параметр	Номер сгенерированного массива				Среднее значение
		1 (упор. по возрастанию)	2 (упор. по убыванию)	3 (случ. порядок элементов)	4 (случ. порядок элементов)	
10	compare					
	swap					
	time					
100	compare					
	swap					
	time					
...	...	...	...	...	...	...

Варианты (стр. указаны для книги [2]):

Сортировки методом подсчета:

1. Подсчет сравнений – стр. 87(Алгоритм C).
2. Подсчет распределений – стр. 89 (Алгоритм D).

Сортировки методом выбора:

3. Сортировка выбором.
4. Пирамидальная сортировка – стр. 159 (Алгоритм H).

Сортировки методом вставок:

5. Сортировка вставками.
6. Метод двухпутевых вставок. – стр. 93.
7. Сортировка Шелла – стр. 95 (Алгоритм D).
8. Метод вставки в список – стр. 107 (Алгоритм L).
9. Сортировка с вычислением адреса – стр. 111(Алгоритм M).

Обменные сортировки:

10. Сортировка пузырьком.
11. Шейкерная сортировка.
12. Обменная сортировка со слиянием (сортировка Бэтчера) – стр. 123 (Алгоритм M).
13. Обменная сортировка с разделением (быстрая сортировка) – стр. 127 (Алгоритм Q).
14. Обменная поразрядная сортировка – стр. 137(Алгоритм R).

Сортировка посредством слияния:

15. Сортировка слиянием.
16. Сортировка методом естественного двухпутевого слияния– стр. 175 (Алгоритм N).

17. Сортировка методом простого двухпутевого слияния – стр. 177 (Алгоритм S).
18. Сортировка посредством слияния списков – стр. 179 (Алгоритм L).

Сортировка методом распределения (алгоритмы поразрядной сортировки):

19. Поразрядная сортировка списка (Алгоритм R, H) – стр. 188.

Сортировки, объединяющие возможности нескольких сортировок:

20. Timsort – гибридный алгоритм сортировки, сочетающий в себе сортировку вставками и слиянием.
21. Introsort – интроспективная сортировка, которая использует быструю сортировку и переключается на пирамидальную сортировку.

2.4. Самостоятельная работа «Внешняя сортировка»

Реализовать алгоритм внешней сортировки одном из методов:

- естественная сортировка (метод естественного слияния).
- сортировка методом двухпутевого сбалансированного слияния.
- сортировка методом n-путевого слияния.
- многофазная сортировка (Фибоначчиевая).

2.5. Самостоятельная работа «Сортировка за линейное время»

1. Реализовать алгоритм «карманная сортировка» («сортировка вычерпыванием») для чисел, лежащих на отрезке [0,1].
2. Реализовать алгоритм создания словаря – «лексикографическая сортировка» кортежей. ***Зам.*** *Реализовать алгоритм таким образом, чтобы регистр букв не влиял на порядок слов в словаре.*

2.6. Самостоятельная работа «Анализ алгоритмов сортировки»

Для выполнения работы рекомендуется ознакомится с главами 2 (п. 2.3), 4(п. 4.1, 4.2), 7, 8, 9 [1].

Задачи:

Вариант 1

1. Для набора данных {3,19,9,5,12,8,7,4,21,2,6,11} проиллюстрируйте действие процедуры пошагово алгоритм быстрой сортировки.
2. Сортировку вставкой можно представить в виде рекурсивной последовательности следующим образом. Чтобы отсортировать массив $A[1..n]$, сначала нужно выполнить сортировку массива $A[1..n-1]$, после чего в этот отсортированный массив помещается элемент $A[n]$. Запишите рекуррентное уравнение для времени работы этой рекурсивной версии алгоритма сортировки вставкой.

3. Пусть имеется множество S, состоящее из n целых чисел, и отдельное целое число x необходимо определить, существуют ли во множестве S два элемента, сумма которых равна x. Разработайте алгоритм решения этой задачи, время работы которого возрастало бы с увеличением n как $\Omega(n \lg n)$.

Вариант 1

1. Продемонстрируйте пошагово использование сортировки подсчетом на массиве A = {6,0,2,0,1,3,4,6,1,3,2).
2. Сведения о банковских операциях часто записываются в хронологическом порядке, но многие предпочитают получать отчеты о состоянии своих банковских счетов в порядке нумерации чеков. Чеки чаще всего выписываются в порядке возрастания их номеров, а торговцы обычно обналичивают их с небольшой задержкой. Таким образом, задача преобразования упорядочения по времени операций в упорядочение по номерам чеков – это задача сортировки почти упорядоченных массивов. Обоснуйте утверждение, что при решении этой задачи процедура сортировка вставками превзойдет по производительности процедуру быстрая сортировка.
3. Запишите псевдокод алгоритма бинарного поиска (в итерационном или рекурсивном виде). Докажите, что время работы этого алгоритма в наихудшем случае возрастает как в $\Theta(\lg n)$.

Контрольные вопросы:

1. Дайте определение частичного и полного порядка. Приведите примеры множеств, где частичный / полный порядок существует / не существует.
2. Сформулируйте задачу сортировки.
3. Перечислите типы сортировок, охарактеризуйте их.
4. Перечислите категории внутренних сортировок, охарактеризуйте их, приведите временную и емкостную оценки алгоритмов.
5. Перечислите категории внешних сортировок, охарактеризуйте их, приведите оценки алгоритмов.
6. Приведите математическое решение оценки сложности сортировки вычерпыванием («карманной»).
7. Запишите на псевдокоде алгоритм лексикографической сортировки. В чем особенность сортировки кортежей?
8. Какие еще сортировки существуют?
9. Охарактеризуйте подход к решению задач «метод разделяй и властвуй». Приведите примеры.
10. Как бы вы изменили процедуру быстрой сортировки для сортировки в невозрастающем порядке?
11. Чему равно время работы быстрой сортировки в случае, когда все элементы массива A одинаковы по величине?

ЛАБОРАТОРНАЯ РАБОТА № 3
ЛИНЕЙНЫЕ СТРУКТУРЫ ДАННЫХ

3.1. Понятие структуры данных

Структура данных – это систематизированный способ организации данных и доступа к ним.

Требования, предъявляемые к структурам данных:

– Эффективность представления,

– Экономное расходование памяти,

– Быстрый доступ к элементам структуры.

3.2. Массив

Массив – это структура данных для представления множества элементов, однотипных по структуре и способу использования.

Массивы относятся к структурам данных со случайным доступом: для выделения некоторой компоненты к имени массива добавляется индекс (значение специального типа), который можно вычислить, потому элементы массива иногда называют переменным с индексами.

К базовым операциям над массивами относятся поиск и выборочное изменение отдельных компонент. Оценим трудоемкость базовых операций в предположении, что в массиве n элементов.

1. поиск элемента, равного x
   – В произвольном массиве $T(n) = \Theta(n)$.
   – В отсортированном массиве $T(n) = \Theta(\log n)$,
2. поиск максимального (минимального) элемента.
   – В произвольном массиве $T(n) = \Theta(n)$.
   – В отсортированном массиве $T(n) = \Theta(1)$.

Зам. *Основные операции над массивами не предполагают включения новых элементов или исключения элементов*.

3.3. Списки

Список – конечная последовательность элементов, порядок которых определяется с помощью ссылок.

Линейный список – конечная последовательность элементов, структурные свойства которой ограничиваются лишь линейным (одномерным) относительным порядком элементов:

$x_1, \ldots, x_{k-1}, x_k, x_{k+1}, \ldots, x_n, \quad n \geq 0, 1 < k < n.$

К базовым операциям для списка относятся (предполагаем, что в списке n элементов):

– формирование списка, $T(n) = \Theta(n)$.

– просмотр списка, $T(n) = \Theta(n)$.
– поиск некоторого заданного элемента с ключом x, $T(n) = \Theta(n)$.
– поиск максимального (минимального) элемента, $T(n) = \Theta(n)$.
– включение элемента с ключом x, $T(n) = \Theta(1)$.
– исключение элемента, $T(n) = \Theta(1)$.

Наиболее распространенными *способы реализации списка*: реализация в виде двух массивов и в виде последовательно связанных компонентов.

Однонаправленный список – список, в котором предусмотрен жесткий порядок перебора элементов – от первого к последнему.

Двунаправленный список – список, структура которого предусматривает возможность перебора элементов как в прямом, так и в обратном порядке.

Зам. *Способы представления одно- и двунаправленного списков аналогичны.*

Стек – линейный однонаправленный список, в котором все включения и исключения элементов (и обычно всякий доступ) делаются в одном конце списка. Реализуется принцип «последний вошел – первый вышел» (last-in, first-out – LIFO).

Очередь – линейный список, в котором все включения элементов производятся в одном конце списка, а все исключения (и обычно всякий доступ) делаются в другом его конце. Реализуется принцип «первый вошел первый вышел» (first-in, first-out – FIFO).

Дек (очередь с двусторонним доступом) – линейный список, в котором все включения и исключения элементов (и обычно всякий доступ) делаются на обоих концах списка.

3.4. Хэш-таблицы

Если необходимо выполнять только операции поиска, добавления или удаления элемента, то часто применяют так называемое хеширование, а соответствующая структура данных называется ***хеш-таблицей***.

Если выполнять эти операции (для последовательности из n элементов) с использованием массивов или списков, то трудоемкость их в среднем $O(n)$. Если используются поисковые деревья, и если они сбалансированы, то трудоемкость операций $O(\log n)$. Если использовать двоичные векторы, то все эти три операции выполняются за фиксированное время независимо от размера множества, но в этом случае элементами обрабатываемой последовательности могут быть только целые числа из небольшого конечного интервала. В худшем случае поиск в хеш-таблице может занимать столько же времени, сколько поиск в списке, но на практике хеширование достаточно эффективно. При выполнении некоторых естественных условий среднее время выполнения базовых операций есть $O(1)$.

3.5. Самостоятельная работа «Линейные структуры данных»

1. Разработать библиотеку классов для работы со структурами данных: «однонаправленный линейный список», «двунаправленный линейный список», «стек», «очередь», «циклическая очередь», «дек». Реализовать эти структуры данных на массивах и с помощью последовательно связанных компонентов. Реализовать базовые методы для вышеуказанных структур данных. ***Зам***. *Предложите архитектуру классов, в которой выделен общий интерфейс для всех указанных структур данных.*
2. Разработать приложение для моделирования работы системы приема 3 разных специалистов поликлиники с имитацией поминутного мониторинга состояния системы в период с T1 часов до T2 часов. Система состоит из трех кабинетов: K1, K2, K3, очереди Q, стека S и регистратуры R. Люди приходят к конкретным специалистам – D1, D2 и D3, находящимся в соответствующих кабинетах. Пришедший больной ставится в очередь (***Зам***. здесь под очередью будет пониматься структура данных: «список», «стек», «очередь», «дек» – опционально). Если в начале очереди находится человек к специалисту Di и этот специалист свободен, то регистратура R отправляет человека в кабинет Ki, а если кабинет Ki занят, то регистратура R отправляет посетителя в стек и из очереди приглашается следующий человек. Если в вершине стека находится посетитель, и он пришел к Di, который в данный момент свободен, то он отправляется в кабинет Ki. По истечении времени моделирования вывести статистику по работе поликлиники за день. Зам. Пример класса для посетителя поликлиники:

```
public class Person{
  string firstNmae;
  string middleName;
  string lastName;
  int nDoctor; // номер специалиста, задается случайно 1..3
  int numberInTheQueue; // номер в очереди при поступлении
  int time; // время в минутах прихода в поликлинику (можно
использовать другой тип данных)
  int minutes; // в минутах, длительность нахождения в
кабинете, задается случайно при поступлении 1..di
};
```

3.6. Самостоятельная работа «Хэш-таблицы»

1. Реализуйте хеш-таблицу со следующими параметрами. Ключи – целые числа. Метод разрешения коллизий − цепочки переполнения.

Хеш-функция h(x) = x mod M, где M – размер хеш-таблицы − простое число, примерно равное n, где n – количество ключей.

2. Замерьте скорость работы операций вставки, удаления и поиска для n = 1000, 10000, 100000, 1000000 для случайных и отсортированных данных, результаты занесите в таблицу.
3. Реализуйте хеш-таблицу, используя метод хеширования с закрытой адресацией. В качестве M возьмите простое число, превышающее n примерно в 2 раза. Выполните замеры времени работы, результаты занесите в таблицу. Сравните полученные результаты и обоснуйте их.
4. Выполните расчет вычислительной сложности алгоритмов. Во сколько раз увеличивается время работы программы при увеличении размера входных данных в 10 раз? Совпадает ли теоретическая и полученная экспериментально оценки? Если нет, то в чем может быть причина?
5. Предложите хеш-функцию лучше, чем x mod M. Выполните замеры скорости вставки в оба вида хеш-таблиц с этой функцией, сравните с предыдущими результатами.

Контрольные вопросы:

1. Дайте определение понятию «структура данных».
2. Перечислите основные свойства структур данных «односвязный линейный список» / «двусвязный линейный список» / «стек» / «очередь» / «дек»?
3. Напишите псевдокод базовых операций со структурами данных: «односвязный линейный список» / «двусвязный линейный список» / «стек» / «очередь» / «дек».
4. Какие способы реализации структуры данных односвязный линейный список вы знаете?
5. Поясните термины: «хеширование», «хеш-функции» и «хеш-таблицы»?
6. Каким требованиям должна отвечать хеш-функция?
7. Что такое коллизия и как ее избежать?
8. Предположим, что динамическое множество S представлено таблицей с прямой адресацией T длины n. Опишите процедуру, которая находит максимальный элемент S. Чему равно время работы этой процедуры в наихудшем случае?

ЛАБОРАТОРНАЯ РАБОТА № 4
НЕЛИНЕЙНЫЕ СТРУКТУРЫ ДАННЫХ. ДЕРЕВЬЯ

4.1. Деревья. Основные понятия

Древовидная структура обеспечивает естественную форму организации данных, которые иерархичны, т.е. объекты могут быть «выше» или «ниже» относительно друг друга. Иерархическая структура получила широкое применение в файловых системах, графических пользовательских интерфейсах, базах данных, веб-сайтах и т.д.

Терминология древовидной структуры данных основана на генеалогическом древе, в котором для описания взаимоотношений между объектами используются понятия:«родитель»,«сын», «предок», «потомок».

Деревом называется связный граф без циклов.

Корневое дерево – это ориентированный граф, который удовлетворяет следующим условиям:

- имеется в точности один узел, в который не входит ни одна дуга (корень);
- в каждый узел, кроме корня, входит ровно одна дуга;
- из корня имеется путь каждому узлу.

Бинарное дерево – это упорядоченное корневое дерево у каждой вершины которого имеется не более двух сыновей.

В бинарном дереве каждый сын произвольного узла определяется либо как левый сын, либо как правый сын.

Полным бинарным деревом будем называть такое упорядоченное корневое дерево, в котором

- каждая вершина имеет не более двух сыновей;
- заполнение дерева осуществляется от корня к листьям по уровням;
- заполнение уровней производится слева направо.

Бинарное дерево называется ***деревом поиска, (бинарным поисковым деревом)*** если оно организовано так, что для каждого узла *v* справедливо утверждение: все ключи в левом поддереве узла *v* меньше ключа узла *v*, а все ключи в правом поддереве больше.

Зам. *Из определения следует, что в поисковом дереве нет двух узлов с одинаковыми ключевыми значениями.*

Примеры поисковых деревьев: идеально-сбалансированное, сбалансированное (АВЛ), красно-черное, Б-дерево.

4.2. Самостоятельная работа «Бинарное поисковое дерево»

1. Реализовать классы для работы со структурой данных «Бинарное поисковое дерево». ***Зам***. *Предполагается, что у пустого дерева количество вершин равно нулю, а высота пустого дерева равна -1*.

1) Реализовать базовые операции:
- добавление элемента в дерево,
- поиск элемента в дереве,
- удаление элемента из дерева.

2) Для вывода информации предусмотреть возможность 3х способов обхода дерева:
- прямой,
- обратный,
- симметричный.

3) Разработать методы для определения: высоты и глубины узла.

2. Разработать приложение для работы с бинарным поисковым деревом и визуализировать графически построенное дерево. При выполнении операций добавления и удаления узла перерисовывать дерево.
3. По вариантам решить задачу из сборника задач [4, стр.9], тема «Бинарные поисковые деревья».

4.3. Самостоятельная работа «Сбалансированные поисковые деревья»

1. По вариантам, разработать класс для структуры данных
   - АВЛ дерево,
   - красно-черное дерево,
   - 2-3 дерево,

   унаследовав его от бинарного поискового дерева, разработанного при выполнении самостоятельной работы «Бинарное поисковое дерево». Переопределить операции добавления и удаления узла дерева с учетом восстановления инварианта.
2. *Разработать класс и базовые операции для структуры данных Б-дерево. Разработать приложение, демонстрирующее возможности использования этого класса. Разработать метод для визуализации Б-дерева при выполнении базовых операций.

4.4. Самостоятельная работа «Анализ алгоритмов»

Для выполнения работы рекомендуется ознакомится с главами 12, 13, 18 [1].

Задачи:

Вариант 1

1. Изобразите бинарное дерево поиска высотой 2, 3, 4, 5 и 6 для множества ключей {1,4,5,10,16,17,21}.
2. Если у дерева высота h, какое у него может быть минимальное и максимальное число узлов?

3. В чем заключается отличие свойства бинарного дерева поиска от свойства неубывающей пирамиды (=бинарной кучи)? Можно ли использовать свойство неубывающей пирамиды для вывода ключей дерева с n узлами в отсортированном порядке за время $O(n)$? Поясните свой ответ.
4. Разработайте нерекурсивный алгоритм, осуществляющий симметричный обход дерева. ***Зам***. *Имеется простое решение, которое использует вспомогательный стек, и более сложное, но более элегантное решение, которое обходится без стека, но предполагает возможность проверки равенства двух указателей*.
5. Изобразите все корректные Б-деревья с минимальной степенью 2, представляющие множество {1,2,3,4,5}.

Вариант 2

1. Изобразите бинарное дерево поиска высотой 2, 3, 4, 5 и 6 для множества ключей {11,3,5,12,8,18,20}.
2. Если в дереве n листьев, сколько в нем максимум и минимум может быть узлов?
3. Покажите, что, поскольку сортировка n элементов требует в модели сортировки сравнением в худшем случае времени $\Omega(n \lg n)$, любой основанный на сравнениях алгоритм построения бинарного дерева поиска из произвольного списка, содержащего n элементов, также требует в худшем случае времени $\Omega(n \lg n)$.
4. Приведите алгоритм вычисления глубины узлов дерева, имеющий время выполнения $O(n)$, где n - количество узлов дерева.
5. Покажите результат вставки ключей F, S, Q, К, С, L, Н, Г, V, W9 M, R, N, P, A, B, X, Y, D, Z, E в указанном порядке в изначально пустое Б-дерево с минимальной степенью 2. Нарисуйте только конфигурации дерева непосредственно перед выполнением разбиения и окончательный вид дерева.

Контрольные вопросы:

6. Дайте определение понятию «дерево», «корневое дерево».
7. Приведите примеры применения этой структуры данных в IT.
8. Дайте определение понятий «высота узла», «высота дерева», «глубина узла», «уровень узла». Приведите пример.
9. Если в дереве n элементов, какая у него может быть максимальная и минимальная высота?
10. Определите структуры данных для хранения деревьев.
11. Дайте определение понятию «полное бинарное дерево». Какая / какие структуры данных используются для хранения полного бинарного дерева?

12. Дайте определение понятию «d-дерево». Какая / какие структуры данных используются для хранения d-дерева?
13. Какие существуют способы обхода деревьев? Приведите псевдокод этих обходов. Приведите примеры практического использования каждого обхода.
14. Опишите алгоритм для расчета координат узлов деревьев, используемый для визуализации дерева, при котором не будет пересечений между дугами дерева.
15. Дайте определение понятию «бинарное поисковое дерево».
16. Опишите алгоритм построения частотного словаря.
17. Является ли операция удаления «коммутативной» в том смысле, что удаление x с последующим удалением y из бинарного дерева поиска приводит к тому же результирующему дереву, что и удаление y с последующим удалением x? Обоснуйте свой ответ.
18. Какое дерево называется идеально сбалансированным?
19. Какое дерево называется сбалансированным или АВЛ деревом?
20. Приведите пример пошагового построения АВЛ дерева.
21. Нарисуйте АВЛ дерево из 8 узлов. Удалите из дерева узел, который является 3й порядковой статистикой. Пошагового продемонстрируйте восстановление инварианта.
22. Какое дерево называется красно-черным деревом?
23. Чему равно наибольшее возможное число внутренних узлов в красно-черном дереве с черной высотой h? А наименьшее возможное число?
24. Приведите пример пошагового построения красно-черного дерева.
25. Нарисуйте красно-черное дерево из 12 узлов. Удалите из дерева узел, который является 4й порядковой статистикой. Пошагового продемонстрируйте восстановление инварианта.
26. Опишите красно-черное дерево с n ключами с наибольшим возможным отношением количества красных внутренних узлов к количеству черных внутренних узлов. Чему равно это отношение? Какое дерево имеет наименьшее указанное отношение, и чему равна его величина?
27. Какое дерево называется Б-деревом? Приведите пример Б-дерева.
28. Почему минимальная степень Б-дерева не может быть равна 1?
29. Чему равно максимальное количество ключей, которое может храниться в Б-дереве высотой h, выраженное в виде функции от минимальной степени дерева t?
30. Как найти минимальный ключ в Б-дереве? Как найти элемент Б-дерева, предшествующий данному?

ЛАБОРАТОРНАЯ РАБОТА № 5
НЕЛИНЕЙНЫЕ СТРУКТУРЫ ДАННЫХ. КУЧИ

5.1. Структура данных «куча». Основные определения

Приоритетными очередями (***кучами***) будем называть такие структуры данных, которые позволяют выполнять две основные операции:

- добавление элемента;
- удаление минимального элемента.

Т.е. для каждого элемента определено некоторое значение-ключ, по которому определяется «минимальность».

Зам. *Кучи наиболее удобны для поиска кратчайших путей в графах.*

Основное свойство (инвариант 1): Элементы в куче организованы таким образом, что приоритет любой вершины не ниже приоритета каждого из ее «сыновей».

Бинарная куча – это полное бинарное дерево, для которого выполняется основное свойство структуры данных куча. ***Зам***. Бинарная куча еще называется «пирамидой» и используется также в пирамидальной сортировке.

Полным d-деревом называется такое дерево, в котором каждая вершина имеет не более *d* сыновей, а заполнение вершин осуществляется в порядке от верхних уровней к нижним, причем, на одном уровне заполнение вершин дерева производится слева направо.

d-куча это полное *d*-дерево для которого выполняется основное свойство кучи (инвариант 1).

Биномиальный лес – это семейство биномиальных деревьев.

Биномиальная куча – это биномиальный лес для которого выполняются:

- основное свойство кучи (*инвариант* 1),
- дополнительным свойством (*инвариантом* 2) биномиальной кучи является требование, чтобы в куче не было двух биномиальных деревьев одинаковой высоты**.**

Куча Фибоначчи – это семейство корневых деревьев, для которых выполняются следующие свойства (инварианты):

- инвариант 1). Каждый узел в куче Фибоначчи удовлетворяет основному свойству кучи: приоритет отца не ниже приоритета каждого из его сыновей.
- инвариант 2). Каждый некорневой узел может потерять не более одного сына при выполнении процедуры cut.
- инвариант 3). В семействе корневых деревьев нет двух деревьев с корнями одинакового ранга.

5.2. Коды Хаффмана

Коды Хаффмана – это широко используемый метод сжатия, присваивающий символам алфавита коды переменной длины, основываясь на вероятностях появления этих символов. Это очень эффективный метод сжатия данных, который, в зависимости от характеристик этих данных, обычно позволяет сэкономить от 20 до 90% объема. Алгоритм Хаффмана универсальный, его можно применять для сжатия данных любых типов, но он малоэффективен для файлов маленьких размеров (за счет необходимости сохранения словаря).

В настоящее время данный метод практически не применяется в чистом виде, обычно используется как один из этапов сжатия в более сложных схемах. Это единственный алгоритм, который не увеличивает размер исходных данных в худшем случае (если не считать необходимости хранить таблицу перекодировки вместе с файлом).

Предполагаем, что мы рассматриваем данные, представляющие собой последовательность символов. В алгоритме Хаффмана используется таблица, содержащая частоты появления тех или иных символов. С помощью этой таблицы определяется оптимальное представление каждого символа в виде бинарной строки (с помощью кода переменной длины). Далее на основании этой таблицы строится дерево кодирования Хаффмана (Н-дерево).

5.3. Самостоятельная работа «Реализация куч»

1. Разработать класс для структуры данных (вариант определяется преподавателем):
   - «d-куча»,
   - «бинарная куча»,
   - «биномиальная куча»,
   - «куча Фибоначчи».
2. Реализовать операции формирования кучи из множества элементов, добавления элемента в кучу, удаление элемента с наибольшим приоритетом из кучи. При выполнении операций все свойства кучи должны восстанавливаться.
3. Визуализировать кучу при выполнении операций. ***Зам.*** *Учитывайте тот факт, что рассматриваемые кучи являются либо деревом, либо образуют лес. Визуализация деревьев была рассмотрена при выполнении лабораторной работы №4.*

5.4. Самостоятельная работа «Алгоритм сжатия Хаффмана»

1. Разработать приложение для сжатия и распаковки текстового файла с помощью алгоритма сжатия по методу Хаффмана через префиксные коды и на основе кодовых деревьев.

5.5. Самостоятельная работа «Структуры данных»

1. В соответствии с вариантом (сборник задач [3, стр.43], тема «Структуры данных»), проанализировать задачу, определить подходящую структуры данных, построить математическую модель решения, разработать алгоритм решения задачи, оценить сложность алгоритма, разработать приложение, реализующее разработанный алгоритм.

5.6. Самостоятельная работа «Анализ алгоритмов»

Для выполнения работы рекомендуется ознакомится с главами 6, 16.3, 19 [1].

Вариант 1

1. Является ли последовательность {23,17,14,6,13,10,1,5,7,12} бинарной кучей?
2. Напишите псевдокод метода слияния бинарных куч.
3. Пошагово сформируйте кучу Фибоначчи, состоящую из 15 элементов. Процесс создания кучи изобразите графически. Изобразите графически процедуру удаления минимального элемента из кучи, подготовленной в п.5.
4. Покажите, что в любой n-элементной бинарной куче на высоте h находится не более $\lceil n/2^{h+1} \rceil$ узлов.
5. Обобщите алгоритм Хаффмана для кодовых слов в троичной системе счисления (т.е. слов, в которых используются символы 0, 1 и 2) и докажите, что с его помощью получается оптимальный троичный код.

Вариант 2

1. Какую структуру данных представляет собой массив, сортируемый с помощью пирамидальной сортировки? Чему равно время работы алгоритма пирамидальной сортировки массива A длины n, в котором элементы отсортированы и расположены в порядке возрастания? В порядке убывания?
2. Покажите, что время работы алгоритма пирамидальной сортировки в наихудшем случае равно $\Omega(n \lg n)$.

3. Создайте реализацию абстрактных классов «стек» и«очередь» с использование класса «куча».
4. Пошагово сформируйте кучу Фибоначчи, состоящую из 15 элементов. Процесс создания кучи изобразите графически. Изобразите графически процедуру удаления минимального элемента из кучи, подготовленной в п.5.
5. Определите оптимальный код Хаффмана для представленного ниже множества частот, основанного на первых восьми числах Фибоначчи. a:1 b:l c:2 d:3 e:5 f:8 g:13 h:21. Попытайтесь обобщить ответ на случай первых *n* чисел Фибоначчи.

Контрольные вопросы:

1. Дайте определение понятию куча. Сформулируйте основное свойство кучи. Приведите примера использования этой структуры данных.
2. Чему равно минимальное и максимальное количество элементов в куче высотой *h*? Определите это значения для изученных куч.
3. Дайте определение понятию «бинарная куча». Запишите объявление классов для этой структуры данных.
4. Опишите базовые операции над структурой данных «бинарная куча». Запишите их с помощью псевдокода.
5. Дайте определение понятию «d- куча». Запишите объявление классов для этой структуры данных.
6. Опишите базовые операции над структурой данных «d-куча». Запишите их с помощью псевдокода.
7. Дайте определение понятию «биномиальная куча». Запишите объявление классов для этой структуры данных.
8. Является ли массив с отсортированными элементами неубывающей биномиальной кучей?
9. Опишите базовые операции над структурой данных «биномиальная куча». Запишите их с помощью псевдокода.
10. Дайте определение понятию «куча Фибоначчи». Запишите объявление классов для этой структуры данных.
11. Опишите базовые операции над структурой данных «куча Фибоначчи». Запишите их с помощью псевдокода.
12. Опишите алгоритм визуализации кучи.
13. Используя алгоритм сжатия Хаффмана, запишите в бинарном виде закодированную строку для фразы «data structures».
14. Алфавит содержит 7 букв, которые встречаются с вероятностями 0,4; 0,2; 0,1; 0,1; 0,1; 0,05; 0,05. Осуществите кодирование по методу Хаффмана.

ЛАБОРАТОРНАЯ РАБОТА № 6
ОСНОВНЫЕ АЛГОРИТМЫ НА ГРАФАХ

6.1. Графы. Основные определения

Граф G = (V, E) состоит из непустого множества узлов (вершин) V и множества пар узлов E. Будем предполагать, что |V| = n и |E|=m.

Если множество E представлено в виде упорядоченных пар узлов (*v*, *w*), то граф называется *ориентированным* (***орграфом***), а упорядоченная пара узлов (*v*, *w*) называется *дугой*, в противном случае граф называется ***неориентированным***, а пара узлов называется *ребром*.

Путем *(в ориентированном графе*) называется последовательность вершин вида $v_1, v_2, \dots, v_{k-1}, v_k$, где $(v_i, v_{i+1}) \in E$, $i = 1, \dots, k-1$.

Путь называется ***простым***, если все узлы различны. Для *неориентированного графа* аналогичное понятие – ***цепь***.

Цикл – это простой путь, который начинается и заканчивается в одном и том же узле.

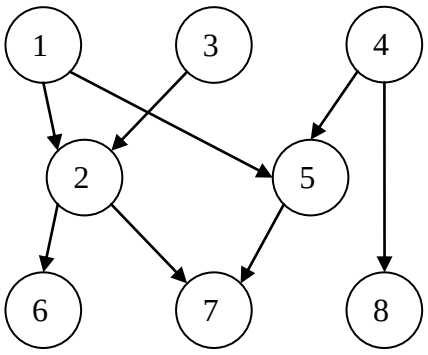
Граф называется ***связным***, если для любой пары вершин *существует простой путь*, их соединяющий; в противном случае, граф называется ***несвязным***.

Если *v* – некоторая вершина графа, то максимальное количество вершин, в которые существует путь из *v* порождают ***компоненту связности графа***. Максимальный подграф, в котором между любой парой вершин существует путь как в одну, так и в другую сторону, называется ***сильносвязной*** компонентой орграфа.

Граф называется ***двудольным***, если множество его вершин можно разбить на два подмножества (доли) таким образом, что каждое ребро соединяет только вершины различных подмножеств.

Эйлеров контур в орграфе – это такой замкнутый путь, который проходит ровно один раз по каждой дуге. ***Эйлеров цикл*** в графе – это такой замкнутый маршрут, который проходит ровно один раз по каждому ребру.

6.2. Четыре способа представления графов в памяти компьютера.



1) Классическим способом задания графа является ***матрица смежности*** *A* размера $n \times n$, где

– $A[v, w] = 1$, если существует ребро (дуга), идущее из вершины *v* в *w*,

– $A[v, w] = 0$, противном случае.

Преимущество такого представления заключается в том, что за $O(1)$ можно дать

ответ на вопрос «существует ли ребро (дуга) (*v*, *w*)?». *Недостаток* заключается в том, что объем требуемой памяти вне зависимости от количества ребер (дуг) составляет n^2.

Матрица смежности

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	1	0	0	1	0	0	0
2	0	0	0	0	0	1	1	0
3	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	1
5	0	0	0	0	0	0	1	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

Матрица инцидентности

	1,2	**1,5**	**2,6**	**2,7**	**3,2**	**4,5**	**5,7**	**4,8**
1	1	1	0	0	0	0	0	0
2	-1	0	1	1	-1	0	0	0
3	0	0	0	0	1	0	0	0
4	0	0	0	0	0	1	0	1
5	0	-1	0	0	0	-1	1	0
6	0	0	-1	0	0	0	0	0
7	0	0	0	7	0	0	-1	0
8	0	0	0	0	0	0	0	-1

2) Другим традиционным способом представления графа служит ***матрица инцидентности***.

Это матрица *A* с *n* строками, соответствующим вершинам, *m* столбцами, соответствующим ребрам.

Для орграфа столбец, соответствующий дуге *(v,w)*, содержит:

- 1, в строке соответствующей вершине *v*, содержит
- (–1), в строке, соответствующей вершине *w*, и
- 0, во всех остальных строках,
- например, 2, можно использовать для обозначения петли.

Этот способ задания графа имеет ряд *недостатков*: требует $n \times m$ ячеек памяти, которые в своем большинстве заполнены нулями; ответы на вопросы: «существует дуга (*v*, *w*)?», «к каким вершинам ведут ребра из вершины *v*?» требуют в худшем случае перебора всех столбцов матрицы (*m* шагов).

3) Более экономным в отношении памяти, особенно, когда $m << n^2$, например, $m = O(n)$, является ***метод представления графа с помощью списка пар, соответствующим ребрам***: пара *v*, *w* соответствует ребру (*v*, *w*).

Преимущество: объем памяти в этом случае составляет $O(m)$. *Неудобство* заключается в большом количестве шагов (в худшем случае порядка *m*), необходимом для получения множества вершин, к которым ведут ребра из данной вершины.

Список пар

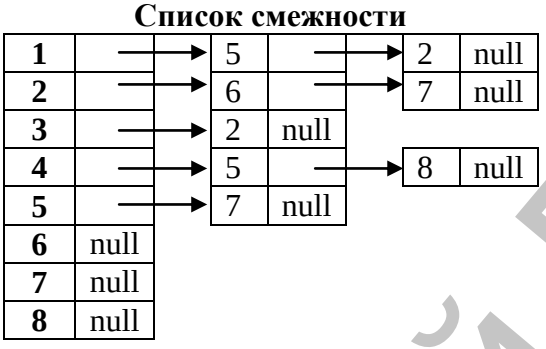
1	1	2	2	3	4	5	4
2	5	6	7	2	5	7	8

4) Во многих ситуациях наиболее приемлемым способом задания графа является структура данных, которую называю ***списками***

смежности. Граф представлен в виде *n* списков смежностей, по одному для каждой вершины.

Массив *graph* – статический массив, элемент *graph* [*i*] – указатель на начало линейного списка вершин, смежных с *i*-ой вершиной; если из вершины *i* нет выходящих дуг, то *graph* [*i*] = *null.*

Преимущество: объем памяти в этом случае *O* (*m* + *n*).



6.3. Обход вершин графа

Обход вершин графа – это базовая процедура исследования графа посредством обращения ко всем его узлам и путям. Эффективным обход считается в том случае, когда обращение к узлам и путям выполняется за линейное временя.

В лабораторной работе рассматриваются два типа прохождения графа: «поиск в глубину» и «поиск в ширину». Они служат основой для разработки ряда эффективных алгоритмов на графах.

Поиск в ширину работает путём последовательного просмотра отдельных уровней графа, начиная с узла-источника *s*. При реализации используется структура данных «очередь». Поиск в ширину обходит связные компоненты графа и в процессе этого обхода определяет соответствующее остовное дерево. Трудоемкость O (*m* + *n*).

Стратегия ***поиска в глубину***, состоит в том, чтобы идти «вглубь» графа, насколько это возможно. Алгоритм поиска описывается рекурсивно. На больших графах поиск в глубину серьёзно нагружает стек вызовов. Если есть риск переполнения стека, используют нерекурсивные варианты поиска, использующие структуру данных «стек» для хранения смежных с текущей вершин. Трудоемкость O (*m* + *n*). Этот алгоритм используется для ряда вычислений в графе, включая поиск пути из одного узла в другой, определение связности графа, и создания остовного дерева связного графа.

6.4. Структура данных «множество»

В некоторых задачах необходимо хранить разбиение какого-то набора объектов на непересекающиеся множества и уметь эффективно выполнять следующие базовые операции:

1. определять принадлежность элемента множеству (имя←НАЙТИ(i));
2. выполнять объединение непересекающихся множеств (ОБЪЕДИНИТЬ (имя1, имя2, имя3)).

Структуру данных, умеющую выполнять такие операции, назовем ***системой непересекающихся множеств***. Объем памяти, необходимый для представления множеств, пропорционален количеству элементов в нем. Время, необходимое для выполнения базовых операций, зависит от структуры данных, выбранной для реализации множества. Возможны реализации на массиве, с помощью списковой структуры, с помощью деревьев.

6.5. Самостоятельная работа «Элементарные алгоритмы работы с графами»

Зам. *Для работы с графами рекомендуется при реализации алгоритмов использовать представление графа в виде списка смежности*.

Варианты:

1. С помощью поиска в ширину / глубину в графе выделить связные компоненты неориентированного графа.
2. С помощью поиска в ширину / глубину в графе определить является ли граф двудольным.
3. С помощью поиска в ширину реализовать алгоритм поиска наименьшего (по числу ребер) пути из заданной вершины до всех оставшихся.
4. Разработать приложение для выполнения топологической сортировки вершин ацикличного орграфа. ***Зам.*** *Граф вводится списком дуг, в памяти хранится в виде списка смежности*.

6.6. Самостоятельная работа «Структура данных «множество»

1. Разработать класс для структуры данных множество. Реализовать базовые операции НАЙТИ, ОБЪЕДИНИТЬ.
2. Варианты для представления множеств: с помощью деревьев, с помощью списков.
3. Визуализировать множество при выполнении базовых операций.

6.7. Самостоятельная работа «Основные алгоритмы работы с графами»

Варианты:

Построение остовного дерева

1. Реализовать алгоритм построения остовного дерева для графа (реализация алгоритм Крускала, с использованием структур данных «куча», «множество»).
2. Реализовать алгоритм построения остовного дерева для графа (реализация алгоритм Прима, с использованием структуры данных «множество»).

Кратчайшие пути из одной вершины

3. Реализовать алгоритма Фолда-Беллмана – алгоритм построения кратчайшего элементарного пути из вершины s во все достижимые из нее вершины. ***Зам***. *При разработке приложения учитывайте требования, которые предъявляет алгоритм к входным данным.*
4. Реализовать алгоритма Дейкстры – алгоритм построения кратчайшего элементарного пути из вершины s во все достижимые из нее вершины.
5. Реализовать алгоритм Дейкстры с использованием структуры данных куча.
6. Реализовать алгоритм Флойда-Варшалла – алгоритм построения кратчайшего элементарного пути из вершины s во все достижимые из нее вершины.

Циклы

7. Разработать приложение для построения эйлерова пути в неориентированном мультиграфе с петлями.
8. Разработать программу, в которой для заданного неориентированного графа $G = (V, E)$ определяется содержит ли он простой цикл. Доказать, что сложность алгоритма $O(|V|)$, не зависимо от $|E|$. ***Зам.*** *Граф вводится списком ребер, в памяти хранится в виде списка смежности*.
9. Разработать приложение для построения эйлерова пути в неориентированном мультиграфе с петлями, если это возможно. ***Зам***. *В теории графов мультиграфом (или псевдографом) называется граф, в котором разрешается присутствие кратных рёбер (их также называют «параллельными»), то есть рёбра, имеющие те же самые конечные вершины*.

Задача о максимальном потоке

10. Реализовать алгоритм Форда-Фалкенсона для построения максимального потока сети.
11. Реализовать алгоритм построения максимального количества путей, не пересекающихся по дугам.

6.8. Самостоятельная работа «Задачи на графах»

1. По вариантам решить задачу из сборника задач [4, стр.61], тема «Графы».

6.9. Самостоятельная работа «Анализ алгоритмов»

Для выполнения работы рекомендуется ознакомится с главами 21-26 [1].

Задачи:

1. При транспонировании графа $G = (V, E)$ мы получаем граф $G^T = (V, E^T)$, где $E^T = \{(v, w) \in V \times V: (w, v) \in E\}$, т.е. граф G^T представляет собой граф G с обратным направлением ребер. Опишите эффективный алгоритм транспонирования графа, как для представления с использованием списков смежности, так и для матриц смежности. Проанализируйте время работы ваших алгоритмов.
1. Пусть (v, w) – ребро минимального веса в графе G. Покажите, что (v, w) принадлежит некоторому минимальному остовному дереву G.
2. Пусть B – матрица инцидентности ориентированного графа G. Поясните, что представляют собой элементы матрицы $B\,B^T$, где B^T – транспонированная матрица B.
3. Предположим, что у нас есть n борцов, между любой парой которых может состояться, (а может и не состояться) поединок, и список r поединков. Разработайте алгоритм, который за время $O(n + r)$ определяет, можно ли разделить всех борцов на две команды так, чтобы в поединках встречались только борцы из разных команд, и если это возможно, то алгоритм должен выполнять такое разделение.

Контрольные вопросы:

1. Сформулируйте основные определения по теме «графы».
2. Опишите методы хранения графов в памяти компьютера.
3. Опишите методы обхода вершин графа – поиск в глубину, поиск в ширину.
4. Что означают понятия «дерево поиска в глубину», «дерево поиска в ширину»?
5. Опишите алгоритм поиск в глубину для графа и для орграфа.
6. Дайте определение понятию «множество». Приведите примера использования этой структуры данных.
7. Опишите способы представления множества с помощью деревьев и с помощью списков.
8. Сформулируйте задачу топологической сортировки. Приведите пример.

9. Дайте определение понятию «остовное дерево». Приведите примеры.
10. Сформулируйте задачу о минимальном остовном дереве. Опишите алгоритм построения минимального остовного дерева. Оцените его сложность.
11. Сформулируйте задачу о кратчайшем пути в графе. Опишите алгоритм решения этой задачи.
12. Сформулируйте задачу о максимальном потоке в графе. Опишите алгоритм решения этой задачи.

ЛАБОРАТОРНАЯ РАБОТА № 7
РАЗРАБОТКА ЭФФЕКТИВНЫХ АЛГОРИТМОВ

7.1. Стратегия метода динамического программирования

Динамическое программирование, как и метод «разделяй и властвуй», рассмотренный при работе над лабораторной работой 2, позволяет решать задачи, комбинируя решения вспомогательных подзадач. ***Зам.*** *Термин «программирование» в данном контексте означает табличный метод, а не составление компьютерного кода*.

В алгоритмах, где применяется метод «разделяй и властвуй» задача делится на несколько независимых подзадач, каждая из которых решается рекурсивно, после чего из решений подзадач формируется решение исходной задачи. Динамическое программирование, напротив, находит применение тогда, когда подзадачи перекрываются, т.е. когда разные подзадачи используют решения одних и тех же подзадач. В этом контексте алгоритм на основе метода «разделяй и властвуй», многократно решая задачи одних и тех же типов, выполняет больше действий, чем необходимо.

В алгоритме динамического программирования каждая подзадача решается только один раз, после чего ответ сохраняется в таблице. Это позволяет избежать одних и тех же повторных вычислений каждый раз, когда встречается данная, уже решенная ранее, подзадача.

Динамическое программирование, как правило, применяется к задачам оптимизации, в которых для получения оптимального решения необходимо сделать определенное множество выборов. После того как каждый *из* выборов сделан, часто возникают вспомогательные подзадачи того же вида. Такая задача может иметь много возможных решений. С каждым вариантом решения можно сопоставить какое-то значение, и нам нужно найти среди них решение с оптимальным (минимальным или максимальным) значением. *Например*, во взвешенном графе между заданной парой вершин существует несколько маршрутов, каждый маршрут характеризуется своей длиной, и нам необходимо найти маршрут кратчайшей длиной.

Стратегия метода динамического программирования – попытка свести рассматриваемую задачу к более простым задачам.

Данная техника находит свое применение, когда все значения оптимальных решений подзадач помещаются в память. Вычисление идет от малых подзадач к большим, и ответы запоминаются в таблице. Одна из клеток таблицы и дает значение оптимального решения исходной задачи.

Благодаря этой простой идее алгоритм, время решения которого экспоненциально зависит от объема входных данных, иногда можно преобразовать в алгоритм с полиномиальным временем работы.

7.2. Четыре этапа процесса разработки алгоритмов по методу динамическое программирование

1. **Задача погружается в семейство вспомогательных подзадач той же природы**. В отличие от метода «разделяй и властвуй» данные подзадачи могут являться зависимыми, т. е. могут пересекаться (различные вспомогательные задачи могут использовать при своем решении оптимальные решения одних и тех же подзадач). Подзадачи должны удовлетворять следующим двум требованиям:

- Подзадачи должны быть более простыми по отношению к исходной задаче. ***Зам***. *Понятие «более простая задача» может в разных случаях пониматься по-разному. Задача может быть проще из-за того, что опущены некоторые ограничения. Она может быть проще из-за того, что некоторые ограничения добавлены.* Однако, как бы ни была изменена задача, если это изменение приводит к решению более простой задачи, то, возможно, удастся, опираясь на эту более простую, решить и исходную.
- Оптимальное решение исходной задачи определяется через оптимальные решения подзадач. ***Зам***. *В этом случае говорят, что задача обладает свойством оптимальной подструктуры, и это один из аргументов в пользу применения для ее решения метода динамического программирования*.

2. **Каждая вспомогательная подзадача решается (рекурсивно) только один раз.** Значения оптимальных решений возникающих подзадач запоминаются в таблице (иногда данный метод называют табличным), что позволяет не решать снова встречавшиеся ранее подзадачи.

3. **Вычисление значения, соответствующего оптимальному решению.** Т.е. для исходной задачи строится возвратное соотношение, связывающее значение оптимального решения исходной задачи со значениями оптимальных решений вспомогательных подзадач (т. е. методом восходящего анализа от простого к сложному вычисляем значение оптимального решения исходной задачи).

4. **Составление оптимального решения на основе информации, полученной на предыдущих этапах.** Часто в задачах помимо значения оптимального решения часто требуется получить и само это решение. Для этого требуется некоторая вспомогательная информация, полученная на предыдущих этапах метода.

7.3. Разработка алгоритма с использованием метода «динамическое программирование»

Задача: Задана группа матриц A_1, A_2,…, A_n. Каждая матрица A_i задана размерами n_i,m_i, $m_i = n_i+1$. Определить, какое минимальное число операций умножения требуется для перемножения n матриц, причем, перемножать можно любые две рядом стоящие матрицы.

Решение: Порядок перемножения всех n матриц неоднозначен. Чтобы устранить неоднозначность, нужно расставить скобки. Порядок расстановки скобок однозначно определит последовательность перемножаемых матриц. Поскольку матричное произведение ассоциативно, то результат не зависит от расстановки скобок, но порядок перемножения может существенно повлиять на время работы алгоритма.

Известно, что для перемножения двух матриц размером [n × m] [m × k] требуется m·n·k операций умножения.

Рассмотрим пример перемножения матриц: $A=A_1(10\times20)\cdot A_2(20\times50)\cdot A_3(50\times1)\cdot A_4(1\times100)$. Способ вычисления их произведения можно полностью определить с помощью скобок пятью разными способами, которые потребуют разного количества операций умножения: $(A_1(A_2(A_3A_4)))$, $(A_1((A_2A_3)A_4))$, $((A_1A_2)(A_3A_4))$, $((A_1(A_2A_3))A_4)$, $(((A_1A_2)A_3)A_4)$.

Например, $(A_1(A_2(A_3A_4)))$ – 125 000 операций умножения,

$(A_1(A_2A_3))A_4$ – 2 200.

Если не использовать принцип динамического программирования, то процесс перебора всех порядков, в которых можно вычислить рассматриваемое произведение матриц с целью минимизации количества операций умножения, имеет экспоненциальную сложность. Динамическое программирование дает алгоритм сложностью O (n^3).

Пусть F[i, j] – минимальное количество операций для перемножения матриц с номерами от i до j.

F[i, j]=min (F[i, k]+F[k+1, j]+ni·nk+1·mj), F[i, i]=0

i ≤ k ≤ j-1

Элементы F[i, j] вычисляются в порядке возрастания разностей индексов, т.е. начинаем с вычисления F[i, i] для всех i, затем F[i, i + 1] для всех i, затем F[i, i + 2] для всех i, и т.д. Матрица F задается верхним треугольником, результат решения задачи соответствует величине F[1, S].

Программная реализация этого алгоритма:

```
for (i = 1; i <= n; i++){
     F[i,i] = 0;
}
for (p = 1; p < s; p++){
     for (i = 1; i <= s-p; i++){
          // infinity - максимальное из возможных
```

```
            // оптимальных значений подзадач
            number = infinity;
            j = i + p;
            for (k = i,kOptim = i; k < j; k++){
                 if (number>(f[i, k] + f[k + 1,j] + ni*mk*mj)){
                      number = (f[i, k] + f[k + 1,j] + ni*mk*mj;
                      kOptim = k;
                 }
            }
            F[i,j] = number;
            K[i,j] = kOptim;
      }
}
```

Используя информацию, полученную в процессе формирования матрицы F (сохраненные значения `k`), можно будет указать оптимальный порядок перемножения.

Для примера $A=A_1(10\times20)\cdot A_2(20\times50)\cdot A_3(50\times1)\cdot A_4(1\times100)$ в результате имеем таблицу, из которой следует что минимальное число операций, требуемое для перемножения данных четырех матриц равно 2 200, оптимальный порядок перемножения матриц: $(A_1(A_2(A_3A_4)))$.

		(m_1=20)	**(m_2=50)**	**(m_3=1)**	**(m_4=100)**
	i / j	**1**	**2**	**3**	**4**
(n_1=10)	**1**	0	10 000 k=1	1 200 k=1	**2 200 k=3**
(n_4=20)	**2**		0	1 000 k=2	3 000 k=3
(n_3=50)	**3**			0	5 000 k=3
(n_4=1)	**4**				0

Значение *k* показывает, где при оптимальной расстановке скобок выполняется разбиение последовательности:

$A_1 A_2 \ldots A_j = (A_1 A_2 \ldots A_k)(A_{k+1} \ldots A_j)$

Таким образом, оптимальное вычисление произведения матриц выглядит как $(A_1A_2\ldots A_{K[1,n]})(A_{K[1,n]+1}\ldots A_n)$. Все предшествующие произведения матриц можно вычислить рекурсивно, поскольку элемент `K[1,K[1,n]]` определяет матричное умножение, выполняемое последним при вычислении $A_{K[1,n]+1}\ldots A_n$. Приведем рекурсивную процедуру вывода оптимального способы расстановки скобок в последовательности матриц $A_1 A_2 \ldots A_j$ по индексам *i* и *j*, и по таблице *K*, полученной в результате работы описанного выше алгоритма. Первоначальный вызов процедуры printOptimal(K, l, *n*) выводит оптимальную расстановку скобок в последовательности $A_1, A_2, \ldots, A_n$.

```
printOptimal(K, i, j){
     if (i == j) {
          print "A",i
```

```
    }
    else print "("
    printOptimal(K, i, K[i,j])
    printOptimal(K, K[i, j] + 1, j)
    print")"
}
```

В рассмотренном примере, вызов процедуры `printOptimal(K, 1, 4)` дает строку $(A_1(A_2(A_3A_4)))$.

7.4. Самостоятельная работа «Анализ алгоритмов»

Для выполнения работы рекомендуется ознакомится с главой 15 [1].

Задачи:

1. Числа Фибоначчи определяются рекуррентным соотношением. Разработайте алгоритм динамического программирования со временем работы *O(n)* для вычисления *n*-го числа Фибоначчи. Изобразите граф подзадач. Сколько вершин и ребер имеет этот граф?
2. Приведите алгоритма рекурсивного и не рекурсивного возведение числа в степень. Оцените их трудоемкость.
3. Изобразите рекурсивное дерево для процедуры «сортировка слиянием», рассматриваемой при выполнении лабораторной работы №2, и работающей с массивом из 16 элементов. Объясните, почему для повышения производительности работы хорошего алгоритма, следующего принципу «разделяй и властвуй», такого как «сортировка слиянием», использование запоминания не даст никакого улучшения производительности.
4. Рассмотрим разновидность задачи о перемножении цепочки матриц, цель которой – так расставить скобки в последовательности матриц, чтобы количество скалярных умножений стало не минимальным, а максимальным. Проявляется ли в этой задаче оптимальная подструктура (Можно ли выполнить первый этап разработки алгоритма по методу динамическое программирование)? ***Зам.*** *Оптимальная подструктура проявляется в задаче в том случае, если в ее оптимальном решении содержатся оптимальные решения подзадач*.
5. Изучите алгоритм определения наидлиннейшей общей последовательности (например, [1, п. 15.4]). Используйте этот алгоритм для определения самой длинной общей подпоследовательность последовательностей {1,0,0,1,0,1,0,1) и {0,1,0,1,1,0,1,1,0).
6. Приведите алгоритм, предназначенный для вычисления самой длинной монотонно неубывающей подпоследовательности данной

последовательности, состоящей из n чисел. Время работы алгоритма должно быть равно O (n^2).

7. Покажите, что при определенных условиях оптимальное решение данной задачи находится как оптимальное решение подзадач. Представим, что вы хотите обменять одну валюту на другую. Вы надеетесь на то, что если вместо непосредственного обмена выполнить несколько промежуточных обменов на другие валюты, то обмен получится более выгодным. Предположим, что можно выполнять обмен n различных валют, пронумерованных 1, 2, ..., n, причем вы начинаете с валюты 1 и хотите закончить валютой n. Для каждой пары валют i и j у вас имеется обменный курс r_{ij}, означающий, что если у вас есть d единиц валюты i, то вы можете получить за них $d\,r_{ij}$ единиц валюты j. Последовательность обменов может вызывать комиссионные сборы, зависящие от количества произведенных обменов. Пусть c_k – комиссионный сбор, который вы должны заплатить за k обменов. Покажите, что, если $c_k = 0$ для всех $k = 1, 2, \ldots, n$, то задача поиска наилучшей последовательности обменов валюты 1 на валюту n демонстрирует оптимальную подструктуру. Затем покажите, что если комиссионные сборы c_k имеют произвольные значения, то эта задача поиска наилучшей последовательности обменов валюты 1 на валюту n такой оптимальной подструктурой может и не обладать.

7.5. Самостоятельная работа «Подходы к разработке эффективных алгоритмов: «разделяй и властвуй», «динамическое программирование»

1. По вариантам решить задачу из сборника задач [4, стр.17], тема «Разработка эффективных алгоритмов».

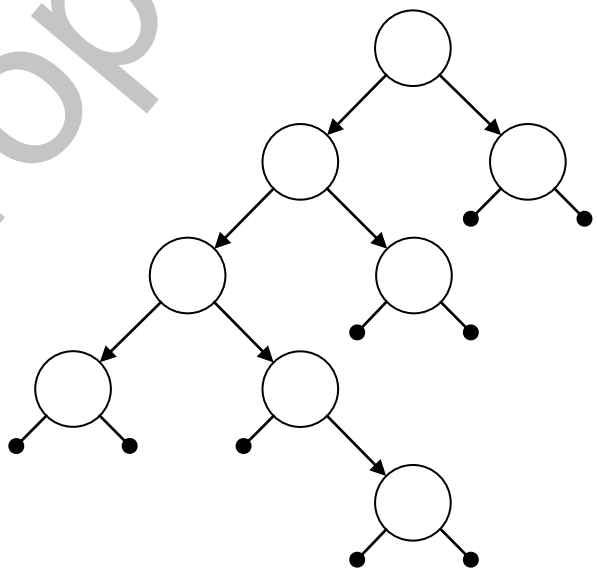
Контрольные вопросы:

1. Сравните подход к решению задач «метод разделяй и властвуй» и «динамическое программирование». Приведите примеры задач, где возможно применение этих подходов.
2. Из каких этапов состоит процесс разработки алгоритмов по методу динамическое программирование.
3. Приведите решение задачи для определения минимального число операций умножения при перемножении n матриц по методу динамическое программирование.
4. Запишите рекурсивный алгоритм для вывода информации о том в какой последовательности надо перемножать матрицы, чтобы решение потребовало минимальное число операций умножения.

ПРИМЕРНЫЕ ВАРИАНТЫ КОНТРОЛЬНЫХ РАБОТ

Контрольная работа №1

1. Выясните, верно ли, что f = O(g) и f = Ω(g) (возможно, верно и то, и другое, тогда f = Θ(g)) для следующих функций:
   а) f(n) = n-100, g(n) = n-200
   б) f(n) = n log n, g(n) = 10n log(10n)
2. Найдите наиболее точную асимптотическую оценку решения рекуррентных уравнений, приведите подробное решение уравнений.
   а) $T(n)=2T(n/8)+n^{1/3}$
   б) T(n)=T(n/3)+T(n/4)+5n
3. Расположите приведенные ниже функции по скорости их асимптотического роста. Ответ обоснуйте.
   а) $2^{\lg n}$ б) n в) $n^{1/\lg n}$ г) lg lg n д) $n^{\lg \lg n}$
4. В массиве *A* [1..*n*] записана возрастающая последовательность целых чисел. Опишите алгоритм, который проверяет, существует ли такой индекс *i*, что *A*[*i*] = *i*. ***Зам.*** *Описание алгоритма может быть на псевдокоде или словесное. Алгоритм должен быть разработан по методу «разделяй и властвуй».*
5. Разместите ключи {25, 14, 5, 7, 11, 3, 17, 19} в узлы бинарного поискового дерева, представленного ниже, так, чтобы свойство бинарного поискового дерева были выполнены.
   1) Определите корень дерева.
   2) Запишите каноническое представление корневого дерева.



6. Пронумеруйте узлы дерева задачи 5, заполните таблицу:

№ узла	1	2	3	4	5	6	7	8
Ключ узла								
Глубина узла								
Высота узла								
Уровень узла								

7. Запишите на псевдокоде алгоритм проверки баланса скобок в арифметическом выражении, заданном в виде строки. ***Зам.*** *При*

реализации алгоритма используйте структуру данных «стек». Скобки могут быть нескольких видов: ( ), { }, [ ].

8. Для данного неотсортированного массива B [1..2n+1] действительных чисел предложите эффективный алгоритм такой перестановки элементов массива B, чтобы массив B был «волнистым множеством». ***Зам.*** *Массив A [1..2n+1] называется «волнистым множеством», если* $A[1] \le A[2] \ge A[3] \le A[4] \ge \ldots \le A[2n] \ge A[2n+1]$.
9. Для отображения данных в табличные индексы используйте хеш-функцию *hashf*(x) = x % 11. Данные вставляются в таблицу в следующем порядке: 11, 13, 12, 34, 38, 33, 27, 22.
   а) Постройте хеш-таблицу методом открытой адресации.
   б) Постройте хеш-таблицу методом цепочек.

Контрольная работа № 2

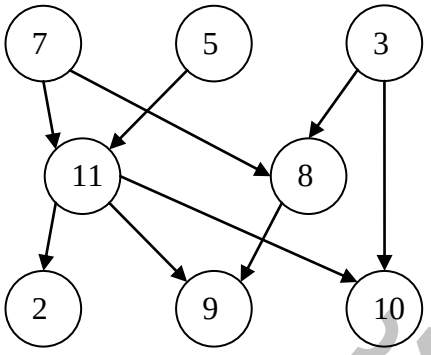
1. Укажите асимптотическую оценку сложности указанных ниже алгоритмов в *худшем* случае.

Операция	**Оценка**	**Операция**	**Оценка**
Сортировка вставками		Сортировка с помощью бинарной кучи	
Добавление элемента в АВЛ дерево		Прямой обход дерева	
Построение d-кучи		Быстрая сортировка	
Сортировка лексикографическая		Шейкерная сортировка	
Удаление элемента из бинарного поискового дерева		Построение частотного словаря	

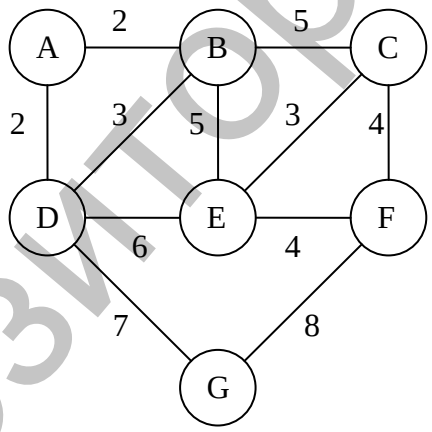
2. Определите верно ли утверждение и кратко *поясните свой ответ*: Дан массив A, состоящий из n чисел. *Утверждение*: построение кучи из n элементов массива A выполнится асимптотически быстрее чем построение красно-черного дерева из тех же элементов.
3. Проиллюстрируйте геометрически пошаговое создание красно-черного дерево из ключей {25, 14, 5, 7, 11, 3, 17, 19}. Добавление узла и восстановление свойств красно-черного иллюстрируйте как два шага.
4. Для дерева задачи 3 напишите псевдокод методов для выполнения необходимых поворотов.
5. Напишите класс для представления графа в виде списка смежности. Напишите псевдокод метода создания графа в виде списка смежности на основе чтения из исходного файла информации о дугах графа.
6. Напишите псевдокод методов, определяющих высоту узлов в бинарном поисковом дереве. ***Зам.*** *Один метод определяет высоту*

узла дерева, второй метод выполняет обход по дереву и вызывает первый метод.

7. Выполните топологическую сортировку для данного графа, заданного в виде списка смежности. ***Зам.*** *Дайте описание алгоритма в псевдокоде или словесное, графически проиллюстрируйте шаги алгоритма.*



8. Для данного графа постройте минимальное остовное дерево, используйте структуры данных «куча», «множество». ***Зам.*** *Дайте описание алгоритма в псевдокоде или словесное, графически проиллюстрируйте шаги алгоритма.*



9. Задана последовательность из k матриц: $A_1(n_1 \times m_1)$, $A_2(n_2 \times m_2)$, …, $A_k(n_k \times m_k)$. Запишите алгоритм для определения минимального числа операций умножения, которое требуется для перемножения этих матриц, в порядке их следования. Продемонстрируйте ваш алгоритм на примере решения задачи для 4 матриц: $A_1(20 \times 5)$, $A_2(5 \times 10)$, $A_3(10 \times 30)$, $A_4(30 \times 50)$. ***Зам***. Используйте метод «динамического программирования» и решение представьте в виде таблицы.

ПРИМЕРНЫЙ ПЕРЕЧЕНЬ ТЕОРЕТИЧЕСКИХ И ПРАКТИЧЕСКИХ ВОПРОСОВ, ВЫНОСИМЫХ НА ЗАЧЕТ

Теоретические вопросы:

1. Предмет теории алгоритмов. Историческое развитие теории алгоритмов и её место среди других математических наук и в естествознании.
2. Понятие информации. Мера информации.
3. Формальное описание задачи. Размерность задачи. Трудоемкость алгоритмов: наилучший случай, наихудший случай, трудоемкость в среднем, усредненная оценка трудоемкости группы операций.
4. Асимптотики O, Ω, Θ. Полиномиальные, псевдополиномиальные и экспоненциальные алгоритмы. Примеры алгоритмов решения задач и оценка их трудоёмкости.
5. Понятие рекуррентного уравнения. Правильные и неправильные рекуррентные уравнения. Полное рекуррентное уравнение. Основные методы решения рекуррентных уравнений: метод итераций и метод рекурсивных деревьев. Оценка решения рекуррентного уравнения: метод подстановок.
6. Теорема о решении рекуррентного уравнения вида $T(n) = a \cdot T\left(n/c\right) + b \cdot n$.
7. Основные подходы к разработке эффективных алгоритмов: «динамическое программирование». Примеры решения задач.
8. Основные подходы к разработке эффективных алгоритмов: метод «разделяй и властвуй». Примеры решения задач.
9. Внутренние сортировки: выбором, вставками, обменные (пузырьковая, шейкерная), быстрая, слиянием, пирамидальная. Оценка трудоёмкости алгоритмов сортировки, используя рекуррентные уравнения.
10. Внешние сортировки. Оценка трудоёмкости алгоритмов, используя рекуррентные уравнения.
11. Алгоритмы поиска элемента, равного X, алгоритмы поиска k-й порядковой статистики. Оценка трудоёмкости алгоритмов.
12. Способы организации базовых структур данных: массив, простой список, мультисписок, стек, очередь. Реализация базовых операций и их трудоёмкость.
13. Технология использования простейших структур данных на примере алгоритма сжатия информации Хафмена.
14. Приоритетная очередь. Бинарная куча, d-куча, Биномиальная куча. Куча Фибоначчи. Реализация базовых операций и их трудоёмкость. Амортизированная (усреднённая) оценка трудоёмкости операции. Примеры решения задач.

15. Система непересекающихся множеств. Различные способы представления системы непересекающихся множеств в памяти компьютера. Реализация базовых операций и их трудоёмкость. Примеры решения задач.
16. Прямая адресация. Хэш-таблицы и хэш-функции. Открытое и закрытое хеширование. Методы разрешения коллизий: метод цепочек, открытая адресация.
17. Методы хранения деревьев в памяти компьютера.
18. Бинарные поисковые деревья. Сбалансированные поисковые деревья: идеально сбалансированные деревья, АВЛ-деревья, красно-черные деревья, 2-3-деревья. Поддержка инвариантов сбалансированности. Реализация базовых операций и их трудоемкость.
19. Графовые модели. Методы хранения графов в памяти компьютера.
20. Алгоритм поиска в глубину в графе и его трудоёмкость. Алгоритм поиска в ширину в графе и его трудоёмкость. Связность, двудольность графа. Выделение сильно связанных компонент ориентированного графа.
21. Маршруты, обладающие заданными свойствами. Подграфы. Топологическая сортировка. Эйлеров цикл.
22. Алгоритмы построения кратчайших маршрутов в графе и их трудоёмкость. Различные подходы к программной реализации алгоритма Дейкстры и их трудоёмкость.
23. Минимальное остовное дерево графа. Алгоритм Прима. Алгоритм Крускала. Трудоёмкость алгоритмов построения минимального остовного дерева.
24. Максимальный поток в сети и его приложения.

Практические задания:

Разрабатываемое в соответствии с вариантом приложение должно удовлетворять требованиям, предъявляемые к приложениям, разрабатываемым в рамках курса.

1. Реализовать 2 алгоритма внутренний сортировки на выбор преподавателя. Вычислить сложность алгоритмов сортировки, используя рекуррентные уравнения. Сравнить сложности алгоритмов при $n \to \infty$. Сравнить 2 алгоритма по числу сравнений элементов, числу обменов, времени работы алгоритмов, когда размер исходного массива элементов n = 2, 10, 100, 1000, 10 000 для случаев: исходная последовательность отсортирована по возрастанию, убыванию, частично отсортирована, неотсортирована. ***Зам.*** *Список сортировок: выбором; подсчетом сравнений, подсчетом распределений; вставками, методом двухпутевых вставок, Шелла, вставки в список, с вычислением адреса; пузырьком, шейкерная, быстрая, Бэтчера, обменная поразрядная; пирамидальная; слиянием, методом*

естественного двухпутевого слияния, методом простого двухпутевого слияния, посредством слияния списков; поразрядная сортировка списка.

2. Разработать приложение, которое выполняет сортировку объектов определенной структуры по определенному ключу. Для сортировки объектов использовать заданный алгоритм внутренний сортировки элементов. Определить число сравнений элементов, число обменов для заданного количества объектов *n*. ***Зам.*** *Вид сортировки, структура элементов, ключ для сортировки определяется преподавателем. Список сортировок см. предыдущем задании. Структура объекта*:

```
public class Person{
  string firstNmae;
  string middleName;
  string lastName;
  int age;
  double height; // в метрах
  double weight; // в килограммах
};
```

3. Реализовать алгоритм внешней сортировки методом, определяемым преподавателем: методом естественного слияния, методом двухпутевого сбалансированного слияния, методом n-путевого слияния, методом многофазной сортировки (Фибоначчиевая сортировка). ***Зам.*** *Дополнительно разработать метод-утилиту, который создает исходный файл/файлы заданного большого размера*.
4. Разработать алгоритм сортировка вычерпыванием. Входные данные должны иметь случайные значения в диапазоне от 0 до 1, и должны быть сгенерированы на основе равномерного закона распределения. Доказать, что сложность алгоритма сортировки *O(n)*.
5. Реализовать алгоритм лексикографической сортировки.
6. Разработать алгоритм поиска *k*-й порядковой статистики на основе идей алгоритма быстрой сортировки.
7. Реализовать 2 алгоритма поиска максимального и минимального элементов в неупорядоченном массиве размера *n*. Один алгоритм с использованием техники «разделяй и властвуй», другой – без ее использования. Оценить данные алгоритмы по числу операций сравнения при различных размерах массива.
8. Разработать приложение, которое для данных строковых величин *A* и *B* определяет минимальное число вставок в строку *B*, замен и удалений символов в строке *A*, требуемое для преобразования строки *A* в строку *B*. Дополнительно вывести все промежуточные шаги преобразования строки *A* в строку *B*. ***Зам.*** *Использовать технику «динамическое программирование».*

9. Разработать приложение, демонстрирующее возможность использования структуры данных «очередь», реализация которой выполнена с использованием двух объектов класса «стек». ***Зам.*** *Использовать собственную реализацию класса «стек».*
10. Разработать приложение, демонстрирующее возможность использования структуры данных «стек», реализация которой выполнена с использованием двух объектов класса «очередь». ***Зам.*** *Использовать собственную реализацию класса «очередь».*
11. Разработать приложение, демонстрирующее возможность использования структуры данных «очередь» / «стек», реализация которых выполнена с использованием объектов класса «однонаправленный список». ***Зам.*** *Использовать собственную реализацию класса «однонаправленный список». Базовые операции добавления, удаления элемента из структуры данных «очередь» / «стек» должны выполняться за время* $O(1)$.
12. Разработайте приложение с классом «стек», в котором реализована возможность использования двух объектов класса «стек» на одном массиве $A[1..n]$. ***Зам.*** *Стеки не переполняются пока суммарное количество элементов в обоих стеках не достигнет* n.
13. Разработать приложение, которое на основе исходной последовательности данных строит «односвязный список» / «стек» / «очередь», а потом, на основе первого объекта строит, второй, аналогичный первому объект, имеющий обратный порядок следования элементов по отношению к исходному. ***Зам.*** *Операции создания и обращения «односвязного списка» / «стека» / «очереди» и их обращения должны выполняться за время* $O(n)$.
14. Разработать приложение, которое на основе исходной последовательности данных строит «стек» / «очередь», а потом, строит второй список из первого, имеющий обратный порядок следования элементов по отношению к исходному. ***Зам.*** *Операции создания и обращения списка и его обращения должны выполняться за время* $O(n)$.
15. Разработать приложение для построения упорядоченного «однонаправленного списка» на основе неупорядоченной последовательности входных данных.
16. Разработать приложение для сжатия и распаковки текстового файла с помощью алгоритма сжатия информации Хафмена.
17. Разработать приложение для нахождения k-го минимального элемента последовательности из n элементов используя при реализации структуру данных «куча». ***Зам.*** *Использовать структуру данных «куча» по выбору преподавателя (бинарная куча, d-куча, биномиальная куча, куча Фибоначчи)*.

18. Разработать программу для моделирования системы управления процедурами посадки и взлета в аэропорту. Любое событие определяется реальным временем наступления события. В программе должны быть реализованы основные операции: добавление события, выбора ближайшего события, удаление прошедшего события. ***Зам.*** *Использовать структуру данных «куча» по выбору преподавателя (бинарная куча, d-куча, биномиальная куча, куча Фибоначчи)*.
19. Разработать программу для подсчета количества различных категорий товаров. Входной информацией является детальный список из *n* товаров. Обосновать выбор способа представления множества. ***Зам.*** *Использовать структуру данных «множество»*.
20. Разработать программу для подсчета количества гласных и согласных букв в тексте. Обосновать выбор способа представления множества. ***Зам.*** *Использовать структуру данных «множество»*.
21. Разработать приложение для установления количества слов из одного текстового файла, присутствующих в другом текстовом файле. Использовать хэш-таблицу, хэш-функцию. Выбрать способ разрешения коллизий. ***Зам.*** *Рекомендуется сначала удалить из обоих файлов различные управляющие символы, а также символы пунктуации и числа*.
22. Разработать приложение, которое определяет информацию о водителях про номеру автомобиля. Реализовать операции добавления, удаления, поиска данных. Использовать хэш-таблицу, хэш-функцию. Выбрать способ разрешения коллизий.
23. Разработать программу для построения частотного словаря слов (слова разделяются в тексте пробелами или знаками пунктуации, слова состоят из букв и цифр, в тексте могут встречаться различные символы). Реализовать методы добавления слова в словарь, поиска слова в словаре, вывода частотного словаря. ***Зам.*** *При сравнении слов регистр букв не учитывать*.
24. Разработать программу для построения частотного словаря символов. Реализовать методы добавления кортежа (набора символов) в словарь, поиска символа в словаре, вывода частотного словаря. ***Зам.*** *При сравнении символов регистр букв не учитывать. Предложить эффективный алгоритм со сложностью O(n)*.
25. Разработать приложение, которое на основе данных лог-файла сервера Apache в формате Common Log Format (CLF), создает и выводит список уникальных посетителей ресурса. Обосновать выбор структуры данных. ***Зам.*** *Каждая строка лог-файла является записью отдельного запроса, состоящего из нескольких полей, разделенных пробелами: IP-адрес посетителя, Запрашиваемый URL, Дата и время запроса, Географическое положение клиента, Используемый*

пользователем браузер, Адрес страницы, с которой зашел посетитель и т.д. Используемая операционная система и проч.

Пример записей в лог-файле:

```
127.0.0.1 - - [13/Mar/2016:23:57:49 +0300] "GET
/modules/book/book.css?o1w6hv HTTP/1.1" 200 1036
127.0.0.1 - - [14/Mar/2016:00:07:26 +0300] "GET
/modules/comment/comment.css?o1w6hv HTTP/1.1" 304 -
```

26. Разработать приложение для организации работы с полным бинарным деревом. Реализовать методы добавления элемента, удаления элемента, поиска элемента, визуализации ключей дерева. ***Зам.*** *Узел дерева, кроме ключевого значения, содержит еще дополнительную информацию, например, как в задании 2.*
27. Разработать приложение, которое на основе арифметического выражения, содержащего числа, операции + - * /, скобки ( ), пробелы, строит дерево этого выражения, визуализирует его и вычисляет результат.
28. Разработайте приложение, которое возвращает количество элементов в дереве, меньших заданного значения *x*. Алгоритм должен работать со сложностью $O(log(n))$. ***Зам.*** *Чтобы достичь такой сложности, подумайте, нужно ли внести дополнения в узлы дерева.*
29. Разработать приложение, которое для двух заданных узлов дерева находит их ближайшего общего предка. ***Зам.*** *Достаточно получить сложность* $O(log^2(n))$*, но при желании можно реализовать и более эффективный алгоритм.*
30. Дополнить приложение для работы с бинарным поисковым деревом методом подсчета листов в этом дереве.
31. Разработать приложение, которое строит новое бинарное поисковое дерево из листьев исходного бинарного поискового дерева.
32. Разработать программу, проверяющую является ли является ли бинарное дерево бинарным поисковым деревом.
33. Разработать приложение, которое строит новое бинарное поисковое дерево из узлов, имеющих равную высоту в исходном бинарном поисковом дереве.
34. Разработать приложение, которое позволяет создавать и визуализировать генеалогическое дерево семьи. Обосновать выбор структуры данных и оценить сложность операций добавления узла, поиска узла, визуализации дерева.
35. Разработать приложение, которое на из исходного АВЛ-дерева / красно-черного деревья / 2-3-деревья (с зависимости от вида дерева, реализованного при выполнении лабораторной работы) пошагово строит новое дерево такого же вида используя обход

исходного дерева прямой / обратный / симметричный, на выбор преподавателя.

36. Разработать программу для преобразования способа хранения графа в памяти: граф вводится списком своих дуг / ребер (выбирается опционально). Информация о нем сохраняется в виде списка смежности / матрицы смежности (выбирается опционально). Из выбранного формата хранения выполняется преобразование в формат хранения, не выбранный на предыдущем шаге. Граф визуализируется двумя способами (на основе двух представлений).
37. Разработайте программу, в которой для заданного неориентированного графа $G = (V, E)$ определяется содержит ли он простой цикл. Доказать, что сложность алгоритма $O(V)$, не зависимо от $|E|$. ***Зам.*** *Граф вводится списком ребер, в памяти хранится в виде списка смежности*.
38. Разработать приложение, в котором для неориентированного графа выделяются связные компоненты на основе поиска в глубину / ширину. ***Зам.*** *Граф вводится списком ребер, в памяти хранится в виде списка смежности*.
39. Разработать приложение, в котором определяется является ли граф двудольным на основе поиска в глубину / ширину. ***Зам.*** *Граф вводится списком ребер, в памяти хранится в виде списка смежности*.
40. Реализовать алгоритм поиска наименьшего (по числу ребер) пути из заданной вершины графа до всех оставшихся. ***Зам.*** *Граф вводится списком ребер, в памяти хранится в виде списка смежности*.
41. Разработать приложение для выполнения топологической сортировки вершин ацикличного орграфа. ***Зам.*** *Граф вводится списком дуг, в памяти хранится в виде списка смежности*.
42. Разработать приложение для нахождения кратчайшего цикла в орграфе.
43. Разработать приложение для определения кратчайших путей из заданной шахматной клетки до всех оставшихся для фигуры (по выбору преподавателя): конь / ферзь / слон / ладья.
44. Разработать приложение для построения эйлерова цикла, если это возможно.
45. Реализовать алгоритм Дейкстры с использованием структуры данных куча.
46. Реализовать алгоритм Крускала построения минимального остовного дерево графа с помощью структур данных множество и куча.
47. Реализовать алгоритм Прима с помощью структуры данных множество.

СПИСОК ЛИТЕРАТУРЫ

1. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. – Москва: Вильямс, 2013. – 1328 с.
2. Кнут, Д. Искусство программирования, том 3. Сортировка и поиск: учебное пособие в 3 томах. – 2-е изд. – М.: «Вильямс», 2007. – С. 824.
3. Котов В. М. Алгоритмы и структуры данных: учебное пособие / В. М. Котов, Е. П. Соболевская, А. А. Толстиков. – Минск: БГУ, 2011. – 267 с.
4. Волчкова, Г. П. Сборник задач по теории алгоритмов для студентов физико-математических специальностей БГУ / Г. П. Волчкова, В. М. Котов, Е. П. Соболевская. – Минск: БГУ, 2004. – 59 с.
5. Вирт, Н. Алгоритмы и структуры данных. – Санкт-Петербург: Невский Диалект, 2001. – 351 с.
6. Ахо, А. В. Структуры данных и алгоритмы: учебное пособие / А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман. – Москва: Вильямс, 2000. – 384 с.
7. Касьянов, В. Н. Графы в программировании: обработка, визуализация и применение. – Санкт-Петербург: БХВ, 2003. – 1104 с.
8. Топп, У. Структуры данных в C++ / У. Топп, У. Форд. – Москва: ЗАО Изд-во «Бином», 1999. – 816 с.
9. Емеличев, В. А. Лекции по теории графов / В. А. Емеличев, О. И. Мельников, В. И. Сарванов, Р. И. Тышкевич. – Москва: Наука, 1990. – 383 с.
10. Липский, В. Комбинаторика для программистов. – Москва: Мир, 1988. – 214 с.

Учебное издание

КАЗАНЦЕВА Ольга Геннадьевна

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Методические рекомендации
по выполнению лабораторных работ

Технический редактор *Г.В. Разбоева*
Компьютерный дизайн *Т.Е. Сафранкова*

Подписано в печать .2016. Формат 60x84 $^{1}/_{16}$. Бумага офсетная.
Усл. печ. л. 2,96. Уч.-изд. л. 2,38. Тираж экз. Заказ .

Издатель и полиграфическое исполнение – учреждение образования
«Витебский государственный университет имени П.М. Машерова».

Свидетельство о государственной регистрации в качестве издателя,
изготовителя, распространителя печатных изданий
№ 1/255 от 31.03.2014 г.

Отпечатано на ризографе учреждения образования
«Витебский государственный университет имени П.М. Машерова».
210038, г. Витебск, Московский проспект, 33.