МЕТОДЫ ПРОГРАММИРОВАНИЯ

♦ ИЗДАТЕЛЬСТВО ТГТУ ♦

УДК 004.42
ББК çç973.26-018.2я73-5
К90

Утверждено Редакционно-издательским советом университета

Р е ц е н з е н т

Кандидат технических наук, доцент,
заведующий кафедрой САПР
И.В. Милованов

С о с т а в и т е л и:

Ю.В. Кулаков,
В.Н. Шамкин

К90 Методы программирования : программа, метод. указ. и контр.

вопросы / сост. : Ю.В. Кулаков, В.Н. Шамкин. – Тамбов : Изд-во Тамб. гос. техн. ун-та, 2006. – 32 с. – 100 экз.

Представлены программа, методические указания, вопросы по изучаемым темам, а также список рекомендуемой литературы.

Предназначены для студентов специальности "Комплексное обеспечение информационной безопасности автоматизированных систем".

УДК 004.42
ББК ç973.26-018.2я73-5

Министерство образования и науки Российской Федерации

ГОУ ВПО "Тамбовский государственный технический университет"

МЕТОДЫ ПРОГРАММИРОВАНИЯ

Программа, методические указания
и контрольные вопросы для студентов специальности
"Комплексное обеспечение информационной безопасности
автоматизированных систем"

Тамбов
Издательство ТГТУ
2006

Учебное издание

МЕТОДЫ ПРОГРАММИРОВАНИЯ

Программа, методические указания и контрольные вопросы

С о с т а в и т е л и:
КУЛАКОВ Юрий Владимирович,
ШАМКИН Валерий Николаевич

Редактор О.М. Я р ц е в а
Инженер по компьютерному макетированию М.Н. Р ы ж к о в а

Подписано в печать 17.10.2006.
Формат 60 × 84/16. Бумага газетная. Гарнитура Times New Roman.
1,9 уч.-изд. л. Тираж 100 экз. Заказ № 553

Издательско-полиграфический центр
Тамбовского государственного технического университета
392000, Тамбов, Советская, 106, к. 14

ВВЕДЕНИЕ

Целью дисциплины "Методы программирования" являются обучение студентов принципам построения и анализа алгоритмов, развитие логического мышления, формирование научного мировоззрения и привитие склонности к творчеству. При изучении этого курса используются знания, полученные студентами по дисциплинам "Математический анализ", "Теория вероятностей" и "Языки программирования высокого уровня". Знания и практические навыки, полученные в курсе "Методы программирования", используются обучаемыми в научных дисциплинах, а также при выполнении курсовых и дипломных работ.

Задачей дисциплины "Методы программирования" является преподавание основ: структур данных; оценки сложности алгоритмов; алгоритмов сортировки; алгоритмов поиска; алгоритмов на графах; алгоритмов генерации случайных последовательностей; алгоритмов генерации подстановок.

В результате изучения дисциплины студенты *должны иметь представление*: о способах оценки сложности работы алгоритмов, о возможности модификации алгоритмов с учетом конкретных практических задач; *знать*: принципы, лежащие в основе алгоритмов сортировки и поиска информации, принципы хранения и обработки информации в алгоритмах сортировки, поиска и алгоритмах на графах, методы генерации случайных последовательностей и подстановок; *уметь*: формулировать задачу и использовать для ее решения известные методы, применять полученные знания к различным предметным областям, реализовывать алгоритмы на языках программирования высокого уровня, выбирая структуры данных для хранения информации; *иметь навыки*: написания и отладки программ, реализующих алгоритмы сортировки, поиска и алгоритмы на графах, получения эмпирических оценок трудоемкости алгоритма.

Учебным планом на изучение дисциплины отводится 200 часов, из которых примерно половину составляют аудиторные занятия, а остальное время занимает самостоятельная работа студентов. В процессе обучения планируется проведение четырех контрольных работ. Дисциплина изучается студентами на протяжении двух семестров, в каждом из которых выполняется одно домашнее задание; в конце первого семестра предусмотрен зачет, а в конце второго – экзамен.

Т е м а 1. СТРУКТУРЫ ДАННЫХ

Программа

1.1. Линейные информационные структуры

Стеки, очереди и деки. Последовательное распределение памяти. Связанное распределение памяти. Циклические списки. Списки с двумя связями. Массивы и ортогональные списки.

1.2. Деревья

Прохождение бинарных деревьев. Представление деревьев в виде бинарных деревьев. Другие представления деревьев.

Методические указания

1.1. Линейные информационные структуры

Начните изучение информационных структур с определения понятий: *информационной структуры*, *структурных отношений*, *списка*, *узла списка*, *указателя* (*связи*) *узла списка*, *пустой связи*, *переменной связи*.

Усвойте понятие такой информационной структуры, которая называется *линейным списком*. Установите перечень операций, которые могут выполняться с линейными списками. Рассмотрите наиболее часто встречающиеся линейные списки: *стеки*, *очереди* и *деки*. Проанализируйте общие свойства и принципиальные отличия этих списков.

Познакомьтесь *с последовательным распределением памяти* при хранении линейных списков и понятием *базового адреса*. Оцените удобство последовательного распределения при работе со стеком. Рассмотрите операции включения элемента в стек, а также исключения элемента из стека. Изучите некоторые ухищрения, необходимые для последовательного представления очереди. Рассмотрите операции включения элемента в очередь, а также исключения элемента из очереди. Уделите внимание проблеме выхода очереди за пределы отведенной для нее памяти и решению этой проблемы "замыканием" очереди в кольцо. Рассмотрите операции включения элемента в стек и в очередь, а также исключения элемента из стека и очереди с учетом возникающих ситуаций: *нехватка* и *переполнение*.

Оцените ситуацию в памяти, когда некоторая программа работает не с единственным списком, а с несколькими списками, размер каждого из которых динамически изменяется. Рассмотрите случай очень хорошего сосуществования двух последовательных списков переменного размера, если им позволено расти навстречу друг другу. Убедитесь в том, что нет способа, который позволяет хранить три или более трех списков переменного размера так, чтобы переполнение возникало, когда суммарный размер всех списков превышает отведенную для них область памяти, и "нижний" элемент каждого списка имеет фиксированный адрес. Рассмотрите

важный особый случай, когда каждый из списков переменного размера является стеком. Познакомьтесь с понятием *перепаковки памяти* при возникшей ситуации переполнения в одном из стеков и возможными способами реализации перепаковки. Изучите алгоритм *G*, который выполняет полную перепаковку памяти, основываясь на изменении размера каждого стека с момента последней перепаковки. Выполните этот алгоритм вручную для некоторой входной информации и проанализируйте полученный результат.

Познакомьтесь *со связанным распределением памяти* при хранении линейных списков. Сопоставьте последовательное и связанное распределение памяти и оцените их преимущества и недостатки. Познакомьтесь с понятием *списка свободного пространства* и рассмотрите операции исключения и включения узлов в этот список. Уделите внимание понятию *пула памяти* (списка свободного пространства) и изучите операцию исключения узла из этого списка с проверкой на переполнение. Изучите метод, позволяющий использовать минимально возможный пул памяти. Рассмотрите несколько наиболее распространенных операций со связанными стеками и очередями. Познакомьтесь с задачей *топологической сортировки* и простым способом ее решения. Рассмотрите машинно-ориентированный алгоритм *T* топологической сортировки, использующий последовательную таблицу и связанные списки. Опробуйте этот алгоритм вручную для некоторой входной информации и проанализируйте полученные результаты.

Познакомьтесь с понятием *циклически связанного списка* и особенностями такого списка. Рассмотрите операции включения и исключения узлов, очистки циклического списка, включения одного циклического списка в другой, расщепления одного циклического списка на два. Рассмотрите задачу *сложения двух многочленов* и алгоритм *A* ее решения, в котором многочлены представлены циклическими списками. Опробуйте этот алгоритм вручную для сложения некоторых многочленов и проанализируйте полученные результаты.

Рассмотрите *линейные списки с двумя связями*. Опишите операции включения и исключения информации в левом конце двусвязного списка и добавьте двойственные операции в правом конце, получаемые на основании симметрии. Убедитесь в том, что полученные операции реализуют все действия для дека общего вида. Рассмотрите схему двусвязного списка с *головным узлом* и выясните главную причину предпочтительности такого представления. Изучите операции исключения и включения узлов для двусвязного списка с головным узлом и оцените простоту их реализации.

Обратите внимание на то, что двумерные массивы и массивы более высокой размерности по существу являются простейшими обобщениями линейных списков. Изучите последовательное распределение памяти при хранении массивов, имеющих полную прямоугольную структуру, а также структуру треугольной матрицы. Рассмотрите способ линейной адресации памяти при плотном *упаковывании* вместе двух треугольных матриц одинакового размера в одну прямоугольную матрицу. Познакомьтесь со связанным распределением памяти для многомерных массивов информации и рассмотрите подробно такое представление на примере *разреженных матриц*. Внимательно изучите операцию *осевого шага* в разреженной матрице, представленной *связанными ортогональными списками*, и алгоритм *S* реализации осевого шага. Опробуйте этот алгоритм вручную для некоторой разреженной матрицы и проанализируйте полученные результаты.

1.2. Деревья

Познакомьтесь с понятиями: *дерева*, *корня дерева*, *поддерева*, *степени узла дерева*, *концевого узла*, *узла разветвления*, *уровня узла*, *упорядоченного* и *ориентированного деревьев*, а также *леса*. Обратите внимание на удобство использования в разговоре о деревьях терминологии *генеалогических* деревьев.

Изучите способы представления деревьев с помощью *графов*, *вложенных множеств*, *вложенных скобок*, *уступчатого списка* и *десятичной системы обозначений Дьюи*. Обратите внимание на то, что, например, всякий прямоугольный массив или алгебраическую формулу можно рассматривать как частный случай древовидной структуры. Познакомьтесь с понятием наиболее важного типа деревьев – *бинарного дерева*. Рассмотрите понятие *прохождения бинарных деревьев*, которое выполняется во всех алгоритмах, работающих с древовидными структурами. Изучите *прямой*, *обратный* и *концевой порядки* прохождения бинарного дерева и рассмотрите алгоритм *T* прохождения бинарного дерева в обратном порядке, который использует стек. Опробуйте этот алгоритм вручную для некоторого бинарного дерева и проанализируйте полученный результат.

Познакомьтесь с достаточно остроумным методом представления бинарных деревьев посредством так называемых *прошитых деревьев*, в соответствии с которым все конечные связи заменены *связями-нитями*. Акцентируйте свое внимание на том, как идут *правые* и *левые* связи-нити. Рассмотрите алгоритм *S* определения *обратного преемника* в прошитом бинарном дереве и обратите внимание, что в данном случае стек не требуется. Попробуйте придумать модификацию этого алгоритма для определения обратного предшественника в прошитом бинарном дереве. Обратите внимание на условия, которым должно удовлетворять пустое прошитое дерево, и рассмотрите представление в ЭВМ дерева как прошитого дерева с головным узлом в соответствии с этими условиями. Проанализируйте преимущества прошитого бинарного дерева с точки зрения его прохождения перед непрошитым.

Убедитесь в том, что прошитые деревья растут почти так же легко, как и обычные деревья, изучив алгоритм *I* присоединения одиночного узла в качестве правого поддерева некоторого узла. Опробуйте этот алгоритм вручную для некоторой входной информации и проанализируйте полученный результат. Поменяв ролями левое и правое поддеревья, получите алгоритм производящий подобным же образом включение влево. Обратите внимание на важный промежуточный метод, использующий *право-прошитые* бинарные деревья. Рассмотрите алгоритм *C* копирования бинарного дерева, опробуйте его вручную и проанализируйте полученный результат.

Изучите способ представления деревьев и лесов в виде бинарных деревьев. Рассмотрите лес из двух деревьев и получите соответствующее ему бинарное дерево. Посредством выполнения этой процедуры в обратном порядке убедитесь в том, что всякому бинарному дереву соответствует единственный лес. Изучите прямой, обратный и концевой порядки прохождения лесов, полученные, исходя из указанного соответствия. Для понимания значения этих трех методов прохождения лесов рассмотрите представление дерева с помощью вложенных скобок и установите связь такого представления с прямым, обратным и концевым порядками прохождения.

Познакомьтесь с *представлениями деревьев при последовательном распределении памяти* и акцентируйте свое внимание на случаях, когда такой способ распределения является наиболее подходящим. Исследуйте один из простых способов представления деревьев и лесов, в котором левые связи явно не указаны, и рассмотрите наиболее интересные свойства такого представления.

Выясните, что для полной спецификации всякого ориентированного дерева или леса достаточно знания только всех *восходящих связей*. Рассмотрите важное приложение этого подхода к построению дерева или леса, называемое *обработкой соотношений эквивалентности*. Изучите алгоритм *E* решения такой задачи, опробуйте его вручную для некоторой входной информации и по полученным выходным данным нарисуйте лес, каждое дерево которого определяет свое множество эквивалентных элементов.

Познакомьтесь с использованием понятия дерева при анализе вычислительных алгоритмов. Рассмотрите *систематический метод обнаружения независимых неизвестных* в задаче определения количества реализаций каждого шага алгоритма. Для лучшего понимания этого метода используйте его при решении конкретного примера и проанализируйте полученные результаты.

Вопросы для самопроверки

1. Что такое информационная структура, структурные отношения, список, узел списка, указатель узла списка, пустая связь и переменная связи?
2. Какой список называется линейным и какие операции могут быть выполнены с линейными списками?
3. Какие специальные названия имеют линейные списки, в которых включение и исключение производятся в первом или последнем узлах?
4. Каким образом организовано последовательное распределение памяти машины при хранении линейного списка?
5. Какие операции и в какой последовательности необходимо выполнить, чтобы поместить новый элемент в стек и исключить элемент из стека?
6. Каким образом организовано представление очереди и какие операции необходимо выполнить для включения элемента в конец очереди и исключения начального узла?
7. Как можно обойти проблему выхода очереди за пределы памяти?
8. Что понимается под ситуациями "переполнение" и "нехватка", и как должны быть описаны операции включения и исключения узлов для стека и очереди с учетом этих ситуаций?
9. Каким образом могут хорошо сосуществовать вместе в одной области памяти два списка переменного размера, и почему это невозможно для трех и более списков переменного размера?
10. Что понимается под перепаковкой памяти, и с какими способами перепаковки Вы знакомы?
11. Каким образом организуется связанное распределение памяти при хранении линейных списков?
12. В чем заключаются преимущества и недостатки последовательного и связанного распределения памяти?
13. Что понимается под списком свободного пространства и пулом памяти?
14. Каким образом организован список свободного пространства, и какие операции необходимо выполнить для включения и исключения узлов при работе с этим списком?
15. В чем заключается метод, позволяющий использовать минимально возможный пул памяти?
16. Каким образом реализуются наиболее распространенные операции со связанными стеками и очередями?
17. Какой смысл вкладывается в понятие топологической сортировки, и как можно решить эту задачу простым способом?
18. Каким образом решает задачу топологической сортировки машинно-ориентированный алгоритм *T*?
19. Что обычно понимается под циклическим списком, и каковы особенности такого списка?
20. Как реализуются операции включения и исключения узлов из циклического списка, очистки циклического списка, включения одного циклического списка в другой циклический список, расщепления одного циклического списка на два циклических списка?
21. Каким образом циклические списки используются в алгоритме *A* сложения двух многочленов?
22. Как организованы линейные списки с двумя связями, и какие действия необходимо выполнить для включения и исключения информации на левом и правом концах таких списков?
23. Почему представление двусвязного списка с головным узлом является более предпочтительным, чем представление такого списка без этого узла?
24. Каким образом организуется последовательное распределение памяти при хранении массивов, имеющих полную прямоугольную структуру и структуру треугольной матрицы?
25. Как можно использовать линейную адресацию при плотном упаковывании вместе двух треугольных матриц одинакового размера в одну прямоугольную матрицу?
26. Каким способом организуется связанное распределение памяти для массивов информации и, в частности, разреженных матриц?

27. Что из себя представляют связанные ортогональные списки, и как такие списки использует алгоритм *S*, реализующий операцию осевого шага в разреженной матрице?

28. Каковы определения понятий дерева, корня дерева, поддерева, степени узла дерева, концевого узла, узла разветвления, уровня узла, упорядоченного и ориентированного дерева, леса?

29. Почему в разговоре о деревьях удобно использовать терминологию генеалогических деревьев?

30. Какие способы представления деревьев Вы знаете?

31. Какое дерево называется бинарным?

32. Что вкладывается в понятие прохождения бинарного дерева, и какие порядки прохождения бинарных деревьев Вам известны?

33. Каким образом организуется представление бинарных деревьев посредством прошитых деревьев?

34. В чем заключается преимущество прошитого бинарного дерева перед непрошитым с точки зрения его прохождения?

35. Каким способом деревья можно представить в виде бинарных деревьев?

36. Каковы прямой, обратный и концевой порядки прохождения лесов, и как эти порядки связаны с представлением лесов с помощью вложенных скобок?

37. Каким образом могут быть представлены деревья при последовательном распределении памяти?

38. Почему для полной спецификации всякого ориентированного дерева или леса достаточно знания только всех восходящих связей, и каким образом это обстоятельство используется в алгоритме *E* обработки соотношений эквивалентности?

39. Как понятие дерева используется в систематическом методе обнаружения независимых неизвестных в задаче определения количества реализаций каждого шага некоторого алгоритма?

Литература: [1, с. 289–460], [6, с. 234–255].

Тема 2. АЛГОРИТМЫ СОРТИРОВКИ

Программа

2.1. Сложность алгоритмов

Понятия модели вычислений и оценки сложности алгоритмов.

2.2. Внутренняя сортировка

Основные понятия и стратегии сортировки. Алгоритмы сортировки вставками, выбором, слиянием, обменная сортировка, распределяющая сортировка. Оценка сложности работы алгоритмов внутренней сортировки.

2.3. Внешняя сортировка.

Основные понятия внешней сортировки. Алгоритмы многофазного и каскадного слияния.

Методические указания

2.1. Сложность алгоритмов

Познакомьтесь с определением понятия *модели вычислений* и рассмотрите различные способы представления такой модели. Усвойте, что *анализ сложности алгоритмов* чрезвычайно важен в программировании для ЭВМ и включает в себя оценку используемого объема памяти, а также время, необходимое для выполнения алгоритма. На примере алгоритма *M* поиска максимального элемента одномерного массива детально изучите только анализ времени выполнения алгоритма, поскольку для этого алгоритма достаточно запоминающего устройства фиксированного объема.

2.2. Внутренняя сортировка

Внимательно изучите понятие *сортировки* и некоторые из наиболее важных ее применений. Определитесь с понятиями *записи*, *файла*, *ключа*, *устойчивой сортировки* и *внутренней сортировки*. Познакомьтесь с главными стратегиями сортировки: *сортировкой вставками*, *обменной сортировкой*, *сортировкой посредством выбора* и *сортировкой подсчетом*.

Начните знакомство с методами внутренней сортировки с сортировки подсчетом и алгоритма *C* ее реализации. Выполните этот алгоритм вручную для некоторых входных данных и проанализируйте полученные результаты. Обратите внимание на то, что записи в этом алгоритме не перемещаются. Оцените время работы этого алгоритма и уясните, что он интересен для нас не эффективностью, а главным образом, своей простотой. Рассмотрите затем другую разновидность сортировки подсчетом, называемую *распределяющим подсчетом*, которая действительно весьма важна с точки зрения эффективности в условиях, когда имеется много равных ключей и все они попадают в небольшой диапазон. Изучите алгоритм *D* сортировки распределяющим подсчетом.

Далее изучите семейство методов сортировки, называемое сортировкой вставками и основанное на том, что перед рассмотрением очередной записи предыдущие записи уже упорядочены. Изучите алгоритм *S* сортировки *простыми вставками*, выполните его вручную для некоторых входных данных и проанализируйте полученные результаты. Оцените время работы этого алгоритма. Познакомьтесь с методом сортировки, называемым *бинарными вставками*, который существенно снижает общее число сравнений для вставляемых элементов. Изучите алгоритм *D* метода *сортировки Шелла с убывающим шагом*, реализующий механизм, который позво-

ляет записям перемещаться большими скачками, а не короткими шажками. Оцените время работы этого алгоритма и сравните его с временем сортировки простыми вставками в различных условиях. Рассмотрите другие пути усовершенствования простых вставок, основанные на тщательном анализе структур данных. Изучите алгоритм *L*, реализующий *вставки в список*, и заметьте, что он не перемещает записи в памяти. Оцените время его работы и экономию времени по сравнению с алгоритмом *S*, полученную за счет дополнительного пространства памяти. Познакомьтесь с методом *сортировки с вычислением адреса*, основной идеей которого является такое обобщение метода вставок в список, чтобы располагать не одним списком, а несколькими. Оцените экономию времени от использования нескольких списков вместо одного.

Внимательно познакомьтесь с методами обменной сортировки, предусматривающей систематический обмен местами между элементами пар, в которых нарушается упорядоченность, пока таких пар не останется. Изучите "*метод пузырька*", являющийся наиболее очевидным способом сортировки, и алгоритм *B* его реализации. Выполните этот алгоритм вручную для некоторых входных данных и проанализируйте полученные результаты. Исследуйте время работы алгоритма *B*, и сравните его быстродействие с быстродействием простых вставок (алгоритм *S*). Рассмотрите возможные усовершенствования "метода пузырька" и сделайте вывод о его достоинствах. Рассмотрите метод *обменной сортировки со слиянием*, который предусматривает подбор для сравнений некоторых пар несоседних ключей, и алгоритм *M* его реализации. Познакомьтесь с доказательством того, что магическая последовательность сравнений и обменов, описанная в алгоритме *M*, действительно сортирует любой файл. *Изучите метод Хоара быстрой сортировки* (обменной сортировки с разделением), в котором результат каждого сравнения используется для определения, какие ключи сравнивать следующими. Обратите внимание на то, что такая стратегия не годится для параллельных вычислений, но может оказаться плодотворной для вычислительных машин, работающих последовательно. Рассмотрите алгоритм *Q* обменной сортировки с разделением и проведите его анализ. Изучите метод *обменной поразрядной сортировки*, основанный на двоичном представлении ключей, и рассмотрите реализующий его алгоритм *R*. Оцените время работы алгоритма обменной поразрядной сортировки и сделайте выводы о его быстродействии.

Рассмотрите еще одно важное семейство методов сортировки, называемое сортировкой посредством выбора, которое основано на идее многократного выбора сначала записи с наименьшим ключом, затем записи с ключом, наименьшим из оставшихся, и так далее, пока не будут выбраны все записи. Изучите алгоритм *S* сортировки посредством *простого выбора*, выполните его вручную для некоторых входных данных и проанализируйте полученные результаты. Оцените время работы этого алгоритма и сравните с быстродействием алгоритма простых вставок (алгоритм *S*), а также с "методом пузырька" (алгоритм *B*). Рассмотрите усовершенствования простого выбора, основанные на том, что информация, полученная после первого шага выбора, используется при последующих выборах. В частности, рассмотрите сортировку *посредством выбора из дерева* и алгоритм *H пирамидальной сортировки*. Проанализируйте эффективность алгоритма *H* в сравнении с сортировкой методом Шелла с убывающим шагом (алгоритм *D*) и быстрой сортировкой Хоара (алгоритм *Q*).

Познакомьтесь с множеством сортировок слиянием, где слияние означает объединение двух или более упорядоченных файлов в один упорядоченный файл. Изучите алгоритм *M* сортировки *двухпутевым слиянием*, выполните его вручную для некоторых входных данных. Проанализируйте этот алгоритм. Затем рассмотрите алгоритм *N* сортировки *естественным двухпутевым слиянием*, проанализируйте его и сравните с быстрой и пирамидальной сортировками. Изучите алгоритм *S* сортировки *простым двухпутевым слиянием*, оцените время его работы и сравните с алгоритмом *N*. Рассмотрите алгоритмы *N* и *S* с точки зрения структур данных и рассмотрите алгоритм *L* сортировки *посредством слияния списков*. Оцените затраты памяти и время работы алгоритмов для связанного и последовательного распределения памяти, проведите сравнительный анализ и сделайте выводы.

Изучите интересный класс методов распределяющей сортировки, являющейся, по существу, прямо противоположной слиянию. Рассмотрите алгоритм *R поразрядной сортировки списка*, использующий связанные структуры данных, и вспомогательный алгоритм *H* сцепления очередей. Выполните такую сортировку вручную для некоторых входных данных и проанализируйте полученные результаты. Оцените время работы алгоритма *R*, сравните с временем работы других алгоритмов, построенных на основе аналогичных предположений (метод сортировки с вычислением адреса и алгоритм *L* сортировки посредством слияния списков), и сделайте выводы.

2.3. Внешняя сортировка

Познакомьтесь с понятием *внешней сортировки*. Обратите внимание на то, что внешняя сортировка в корне отлична от внутренней, т.к. время доступа к файлам на внешних носителях нас жесточайшим образом лимитирует. Уясните, что внешняя сортировка сводится в основном к внутренней сортировке с последующим "внешним слиянием".

Изучите метод *многопутевого слияния и выбора с замещением*, являющийся расширением метода внутренней сортировки, основанного на двухпутевом слиянии – процессе объединения двух упорядоченных последовательностей в одну. Познакомьтесь с понятиями *дерева выбора*, *выбора с замещением* и *дерева "проигравших"*. Изучите процедуру получения начальных отрезков посредством выбора с замещением и алгоритм *R* выбора с замещением.

После того как станет понятным образование начальных отрезков, рассмотрите различные методы распределения отрезков на внешних носителях (лентах) и слияния их до тех пор, пока не получится единственный отрезок. Изучите алгоритм *D* сортировки *многофазным слиянием* с использованием "горизонтального" распределения. Рассмотрите другую основную схему, называемую *"каскадным слиянием"*, и модификацию каскадной сортировки, когда добавлена еще одна лента и почти все время перемотки в течение каскадной сортировки можно совместить. Познакомьтесь еще с одним подходом к сортировке слиянием, называемым *осциллирующей*

сортировкой, когда вместо начального прохода распределения производится переключение то на распределение, то на слияние.

Рассмотрите практические аспекты слияния на лентах. Рассмотрите диски и барабаны как средства для внешней сортировки.

Вопросы для самопроверки

1. Какой смысл вкладывается в понятие модели вычислений?
2. Почему анализ сложности алгоритмов чрезвычайно важен в программировании для ЭВМ и что в себя включает?
3. Каким образом выполняется анализ времени выполнения алгоритма, и что является результатом такого анализа?
4. Что понимается под сортировкой, и каковы ее наиболее важные применения?
5. Какой смысл вкладывается в понятия записи, файла, ключа и устойчивой сортировки?
6. Почему внутренняя сортировка так называется?
7. Какие главные стратегии сортировки Вы знаете?
8. В чем заключается главный смысл сортировки подсчетом?
9. Почему при сортировке подсчетом не происходит перемещение записей?
10. Чем отличается от сортировки подсчетом сортировка распределяющим подсчетом?
11. При каких условиях сортировка распределяющим подсчетом является весьма эффективной?
12. На чем основан метод сортировки, называемой сортировкой вставками?
13. Какова роль сортировки простыми вставками в сортировке бинарными вставками?
14. Какой механизм сортировки реализует метод Шелла с убывающим шагом?
15. Как отличается эффективность метода Шелла от сортировки простыми вставками в различных условиях?
16. Какие пути усовершенствования простых вставок, основанные на тщательном анализе структур данных, Вам знакомы?
17. Какую структуру данных использует алгоритм *L*, реализующий вставки в список, и перемещает ли он записи в памяти?
18. Что является основной идеей метода сортировки с вычислением адреса?
19. Удается ли достигнуть экономии времени от использования нескольких списков в методе сортировки с вычислением адреса вместо одного?
20. Какая стратегия сортировки называется обменной сортировкой?
21. Какой метод является наиболее очевидным способом обменной сортировки?
22. Каково быстродействие "метода пузырька" по сравнению с простыми вставками?
23. В чем могут заключаться возможные усовершенствования "метода пузырька", и каковы его достоинства?
24. Какую идею реализует метод обменной сортировки со слиянием?
25. Почему метод Хоара "быстрой сортировки" называют обменной сортировки с разделением?
26. Какой из методов сортировки годится для параллельных вычислений, а какой – для вычислительных машин, работающих последовательно?
27. Каким образом представляются ключи в методе обменной поразрядной сортировки?
28. Какова оценка быстродействия алгоритма *R* обменной поразрядной сортировки?
29. На чем основано семейство методов сортировки, называемое сортировкой посредством выбора?
30. Какие усовершенствования сортировки простым выбором Вам знакомы?
31. Какова главная идея, положенная в основу множества методов сортировки, называемой сортировкой слиянием?
32. В чем заключается смысл сортировки двухпутевым слиянием?
33. Какая сортировка называется пирамидальной?
34. Какое отношение класс методов распределяющей сортировки имеет к сортировке слиянием?
35. Что Вы понимаете под внешней сортировкой, и чем она отличается от внутренней?
36. Какую роль внутренняя сортировка играет во внешней сортировке?
37. В чем заключаются методы двухпутевого и многопутевого слияния?
38. Почему одна из внешних сортировок называется осциллирующей сортировкой?
39. Какие практические аспекты слияния на лентах Вам знакомы?
40. Какую особенность имеют диски и барабаны как средства для внешней сортировки?

Литература: [1, с. 129–140], [3, с. 13–23, 93–450], [4, с. 12–52, 70–122, 322–351], [5, с. 110–131], [6, с. 246–254].

Тема 3. АЛГОРИТМЫ ПОИСКА

Программа

3.1. Последовательный поиск

Основные понятия. Алгоритмы исчерпывающего поиска.

Поиск в последовательно организованном файле.

3.2. Поиск посредством сравнения ключей

Поиск в деревьях. Оптимальные деревья двоичного поиска. Сбалансированные деревья.

3.3. Хеширование

Понятие хеширования. Разрешение коллизий.

Методические указания

3.1. Последовательный поиск

Познакомьтесь с определениями понятий *поиск*, *ключ*, *таблица* или *файл*, *база данных*, *аргумент поиска*, *удачный* и *неудачный поиск*, *поиск с вставкой*. Обратите внимание на возможные способы классификации методов поиска и взаимосвязь между поиском и сортировкой. Рассмотрите примеры возможных сложных задач поиска и способы сведения их к более простым случаям. Познакомьтесь с историей вопроса поиска и его решения в докомпьютерный период, после появления ЭВМ с небольшой внутренней памятью и только последовательными устройствами типа лент для хранения больших файлов, а также с развитием все большей и большей памяти со случайным доступом.

Рассмотрите наиболее очевидный алгоритм *S последовательного поиска*, заключающийся в том, чтобы, начав с начала, продвигаться, пока не найдешь нужный ключ, и тогда остановиться. Обратите внимание на то, что у этого алгоритма может быть два разных исхода: удачный и неудачный. Проанализируйте время работы алгоритма и обратите внимание на то, что этот способ последовательного поиска, несомненно, знакомый всем программистам, не всегда самый лучший. Рассмотрите очевидное изменение, которое убыстряет работу алгоритма последовательного поиска при условии, что записей не слишком мало. Изучите алгоритм *Q быстрого последовательного поиска* и оцените время его работы. Выполните этот алгоритм вручную для некоторых входных данных и проанализируйте полученные результаты. Усвойте важный ускоряющий принцип, использованный при переходе от алгоритма *S* к алгоритму *Q*. Познакомьтесь с полезным изменением алгоритма последовательного поиска в условиях, когда ключи расположены в возрастающем порядке. Рассмотрите алгоритм *T* реализации *последовательного поиска в упорядоченной таблице*. Заметьте, что если величина ключа с равной вероятностью принимает все возможные значения, то в случае удачного поиска алгоритм, по существу, не лучше алгоритма *Q*. Однако, отсутствие нужной записи алгоритм *T* позволяет обнаружить примерно в два раза быстрее. Обратите внимание на то, что в рассмотренных алгоритмах последовательного поиска в целях удобства используются индексные обозначения для элементов таблицы, но аналогичные процедуры применимы и к таблицам в связанном представлении. Оцените время последовательного поиска для известного и неизвестного распределения вероятностей появления ключей.

3.2. Поиск посредством сравнения ключей

Познакомьтесь с задачей поиска в телефонной книге имени абонента по номеру телефона и ее решением путем предварительного упорядочения файла и дальнейшего быстрого поиска. Изучите *понятие бинарного поиска*, основная идея которого довольно проста, детали же нетривиальны, и для многих хороших программистов не одна попытка написать правильную программу закончилась неудачей. Познакомьтесь с алгоритмом *B* бинарного поиска и иллюстрацией поведения этого алгоритма в случаях, когда разыскивается аргумент, равный имеющемуся в таблице, и когда разыскивается отсутствующий аргумент. Выполните доскональный разбор алгоритма *B*, представив его в виде бинарного дерева решений, и оцените быстродействие алгоритма для удачного и неудачного поиска. Рассмотрите одну важную модификацию бинарного поиска, полученную использованием в нем вместо трех указателей лишь двух: на текущее положение и величину его изменения. Изучите алгоритм *U однородного бинарного поиска*, реализующего упомянутую модификацию. Сравните быстродействие однородного бинарного поиска с быстродействием алгоритма *B* и сделайте выводы. Познакомьтесь далее с *фибоначчиевым поиском*, способным соперничать с бинарным поиском. Для снятия некоторой таинственности с фибоначчиева поиска постройте соответствующее дерево решений и проанализируйте его. Рассмотрите алгоритм *F* фибоначчиева поиска и оцените его быстродействие.

Обратите внимание на то, что ранее рассмотренные методы поиска предназначены, главным образом, для таблиц фиксированного размера, поскольку последовательное расположение записей делает вставки и удаления довольно трудоемкими. Уясните, что эффективный поиск по динамически изменяемой таблице возможен при явном использовании структуры бинарного дерева. Познакомьтесь с алгоритмом *T поиска с вставкой по дереву* и оцените время поиска. Сравните эффективность алгоритма *T* с алгоритмами бинарного поиска, использующими неявные деревья. Изучите алгоритм *D* удаления узла из дерева, поскольку иногда бывает нужно заставить ЭВМ забыть один из элементов таблицы.

Исследуйте проблему нахождения *оптимального дерева* – наилучшего дерева для поиска по таблице, если ключи запрашиваются с данными частотами. Рассмотрите алгоритм *K* нахождения оптимальных бинарных деревьев поиска.

Изучите понятия *высоты дерева*, *сбалансированного дерева* и *показателя сбалансированности*. Познакомьтесь с оценкой, как далеко сбалансированное дерево отклоняется от оптимального. Рассмотрите алгоритм *A поиска с вставкой по сбалансированному дереву* и проанализируйте его быстродействие.

3.3. Хеширование

Познакомьтесь с понятиями *хеширования* (рассеянной памяти), *хеш-функции* и *коллизии*. Изучите два основных типа хеш-функций, один из которых основан на делении, а другой на умножении. Обратите внимание на то, каким двум требованиям должна удовлетворять хорошая хеш-функция. Познакомьтесь с понятием *разрешения коллизий*. Рассмотрите разрешение коллизии *методом цепочек*, состоящее в том, чтобы поддерживать несколько связанных списков, по одному на каждый возможный *хеш-адрес*. Изучите алгоритм *C поиска со вставкой по рассеянной таблице с цепочками* и оцените его быстродействие. Рассмотрите другой способ решения проблемы коллизий, называемый *разрешением коллизий* "*открытой адресацией*" и состоящий в том, чтобы полностью отказаться от ссылок и просто просматривать один за другим различные элементы таблицы, пока не будет найден ключ или свободная позиция. Изучите алгоритм *L поиска со вставкой по открытой рассеянной таблице* и познакомьтесь с результатами экспериментов с линейным опробыванием. Выполните сравнительный анализ изученных методов поиска, чтобы руководствоваться им при выборе наилучшего метода для конкретного приложения.

Вопросы для самопроверки

1. Какой смысл вкладывается в понятия поиск, ключ, таблица или файл, база данных, аргумент поиска, удачный и неудачный поиск, поиск с вставкой?
2. Какой поиск называют внутренним и какой внешним?
3. Чем различаются статический и динамический методы поиска?
4. На каком примере можно показать существующую взаимосвязь между поиском и сортировкой?
5. Как Вы понимаете последовательный поиск?
6. Что подразумевается под удачным и неудачным исходами поиска?
7. Какой важный принцип убыстряет работу алгоритма последовательного поиска при условии, что записей не слишком мало?
8. Что понимается под последовательным поиском в упорядоченной таблице?
9. Какие алгоритмы поиска действуют быстрее в случаях удачного и неудачного поиска?
10. Что Вы понимаете под методами поиска, основанными на линейном упорядочении ключей?
11. Какова основная идея бинарного поиска?
12. Чем отличаются иллюстрации поведения алгоритма бинарного поиска в случаях, когда разыскивается аргумент, равный имеющемуся в таблице, и когда разыскивается отсутствующий аргумент?
13. Как выглядит бинарное дерево решений алгоритма бинарного поиска?
14. В чем заключается важная модификация бинарного поиска, называемая однородным бинарным поиском?
15. Как отличается быстродействие однородного бинарного поиска в сравнении с бинарным поиском?
16. Что обычно понимают под фибоначчиевым поиском?
17. Какова оценка быстродействия фибоначчиева поиска?
18. Для каких таблиц предназначены, главным образом, методы бинарного, однородного бинарного и фибоначчиева поиска?
19. Использование какой структуры данных позволяет осуществить эффективный поиск по динамически изменяемой таблице?
20. В чем заключается основная идея поиска со вставкой по дереву?
21. Какова эффективность алгоритма поиска со вставкой по дереву в сравнении с алгоритмами бинарного поиска?
22. Какое дерево является наилучшим для поиска по таблице, если ключи запрашиваются с данными частотами?
23. По какому алгоритму можно найти оптимальное бинарное дерево поиска?
24. Что обычно понимают под высотой дерева, сбалансированным деревом и показателем сбалансированности?
25. Насколько далеко сбалансированное дерево отклоняется от оптимального?
26. Как работает алгоритм *A* поиска со вставкой по сбалансированному дереву, и каково его быстродействие?
27. Какой смысл вкладывают в понятия хеширования (рассеянной памяти), хеш-функции и коллизии?
28. На каких операциях основаны два основных типа хеш-функций?
29. Каким требованиям должна удовлетворять хорошая хеш-функция?
30. Что обычно понимают под разрешением коллизий?
31. В чем заключается разрешение коллизии методом цепочек?
32. Как работает алгоритм *C* поиска со вставкой по рассеянной таблице с цепочками, и каково его быстродействие?
33. В чем состоит способ решения проблемы коллизий, называемый разрешением коллизий "открытой адресацией"?
34. Какие результаты дают эксперименты с линейным опробыванием для алгоритма *L* поиска со вставкой по открытой рассеянной таблице?
35. Что Вы можете сказать о сравнительном анализе изученных методов поиска, чтобы руководствоваться им при выборе наилучшего метода для конкретного приложения?

Литература: [3, с. 464–650], [4, с. 53–69].

Т е м а 4. АЛГОРИТМЫ НА ГРАФАХ

Программа

4.1. Графы

Основные понятия теории графов. Возможные представления графов в ЭВМ.

4.2. Алгоритмы поиска в невзвешенных графах

Алгоритмы поиска связных и двусвязных компонент в неориентированных графах. Алгоритм поиска сильносвязных компонент в ориентированных графах. Алгоритмы нахождения транзитивного замыкания.

4.3. Алгоритмы поиска для взвешенных графов

Остовные деревья. Алгоритмы нахождение остовного дерева минимального веса, определения кратчайших расстояний между вершинами графа.

Методические указания

4.1. Графы

Изучите основные понятия теории графов: *граф*; *дуга*, *инцидентная вершине*; *вершина*, *коинцидентная дуге*; *степень вершины*; *граничные вершины дуги*; *начало и конец дуги*; *частичный граф*; *подграф*; *частичный подграф*; *смежные дуги и вершины*; *окрестность единичного радиуса вершины*; *взвешенная вершина*; *взвешенная дуга*; *взвешенный граф*; *ориентированные* и *неориентированные графы*; *ребро*; *цепь*; *концевая вершина цепи*; *длина цепи*; *составная*, *сложная* и *простая цепь*; *цикл*; *связный граф*; *компонента связности графа*; *несвязный граф*; *расстояние между вершинами графа*; *диаметр графа*; *путь*; *концевая вершина пути*; *начальная* и *конечная вершина пути*; *длина пути*; *составной*, *сложный* и *простой путь*; *контур*; *сильно связный граф*; *компонента сильной связности*.

Рассмотрите три различные возможные представления графов в ЭВМ, которые отличаются объемом занимаемой памяти и временем доступа к данным: *матрицей смежности*, *матрицей инциденций* и *списком смежности*. Так, матрица смежности обеспечивает быстрый доступ к информации о дугах (ребрах) графа, но если в графе мало дуг (ребер), то эта матрица будет содержать гораздо больше пустых клеток, чем заполненных. Длина списка смежности пропорциональна числу дуг (ребер) в графе, однако при пользовании этим списком время получения информации о дуге (ребре) увеличивается. Заметьте, что выбор наилучшего представления определяется требованиями конкретной задачи. Если в графе много вершин, причем каждая из них связана лишь с небольшим количеством других вершин, список смежности оказывается выгоднее, поскольку он занимает меньше места, а длина просматриваемых списков дуг (ребер) невелика. Если же число вершин в графе мало, то лучше воспользоваться матрицей смежности: она будет небольшой и потери при хранении даже сильно разреженной матрицы будут незначительны. Для закрепления знаний о возможных представлениях графов постройте матрицу смежности, матрицу инциденций и список смежности для заданных ориентированного и неориентированного графов, а также решите обратную задачу.

4.2. Алгоритмы поиска в невзвешенных графах

Работая с графами, часто приходится выполнять некоторое действие по одному разу с каждой из вершин графа. Например, некоторую порцию информации следует передать каждому из компьютеров сети. Подобный *обход вершин графа* можно совершать двумя различными способами. При *обходе в глубину* проход по выбранному пути осуществляется настолько глубоко, насколько это возможно, а при *обходе по уровням* равномерно двигаются вдоль всех возможных направлений. Изучите эти два способа подробно, рассмотрите и проанализируйте алгоритмы их реализации. Для некоторых графов определите вручную порядок посещения узлов при обходе в глубину и по уровням, начиная с заданной вершины.

При решении практических задач (например, планирование дорожного строительства, компьютерной сети и т.п.), формулируемых в терминах графов, бывает важно определить не только число компонент связности неориентированного графа или число компонент сильной связности ориентированного графа, но и сами компоненты. Алгоритмы определения компонент связности и сильной связности графа основаны на использовании матриц связности и сильной связности графов, соответственно. Познакомьтесь с понятиями *матрицы достижимости*, *матриц связности и сильной связности*. Рассмотрите *алгоритм Уоршелла*, вычисляющий матрицу достижимости, на основе которой можно получить матрицу связности или матрицу сильной связности. Используя алгоритм Уоршелла для некоторого ориентированного графа с небольшим числом вершин определите вручную его компоненты сильной связности. Заметьте, что в задачах, возникающих на практике, число вершин графа очень велико и ручной метод выделения компонент является нерентабельным. Рассмотрите алгоритм выделения компонент связности или сильной связности, реализуемый на компьютере.

Познакомьтесь с понятиями *двусвязного графа*, *компоненты двусвязности графа* и *точки сочленения*. Анализ компонент двусвязности (например, компьютерной сети) показывает, насколько она устойчива при разрушениях отдельных узлов. Если граф, образованный компьютерами сети, двусвязен, то сеть сохранит способность функционировать даже при выходе из строя одного из компьютеров. Заметьте, что найденные точки сочленения определяют компоненты двусвязности графа. Точки сочленения можно обнаружить с помощью *метода грубой силы*, удаляя из графа поочередно каждую вершину и пользуясь одним из двух алгоритмов обхода

для проверки связности получаемых подграфов. Рассмотрите также способ обнаружения точек сочленения и компонент двусвязности при единственном обходе графа, когда сохраняется незначительное количество дополнительной информации. Выполните упражнения по поиску компонент двусвязности в заданных графах.

Познакомьтесь с понятием *транзитивного бинарного отношения* и *замыкания транзитивного отношения*. Заметьте, что замыкание по транзитивности, например, коммуникационной сети, представленной ориентированным графом, позволяет определить, существует ли возможность переправить сообщение из одного заданного места в другое. Рассмотрите алгоритм вычисления транзитивного замыкания бинарного отношения. Выполните алгоритм вручную для некоторого бинарного отношения и проанализируйте полученные результаты.

4.3. Алгоритмы поиска для взвешенных графов

Познакомьтесь с понятиями *остовного дерева* и *минимального остовного дерева*. Понятие минимального остовного дерева может использоваться, например, в задаче построения железнодорожной сети, связывающей некоторое число городов. В такой задаче известна стоимость строительства отрезка путей между любой парой городов и требуется найти сеть минимальной стоимости. Изучите *алгоритм Дейкстры-Прима* построения минимального остовного дерева, заключающийся в том, что сначала выбирается произвольная вершина графа и включается в остовное дерево. Затем на каждом шаге рассматривается множество ребер, допускающих присоединение к уже построенной части остовного дерева, и выбирается из них ребро с наименьшим весом. Работа алгоритма заканчивается после того, как в остовное дерево попадут все вершины. Исполните этот алгоритм вручную на конкретном примере и проанализируйте полученный результат. Изучите *алгоритм Крускала* построения минимального остовного дерева, заключающийся в том, что все начинается с пустого дерева, к которому добавляются ребра в порядке возрастания их весов. Процесс добавления ребер продолжается до тех пор, пока не получится набор ребер, объединяющий все вершины графа. Найдите с помощью этого алгоритма минимальное остовное дерево для того же графа, что и при рассмотрении алгоритма Дейкстры-Прима, и сравните полученные результаты.

Рассмотрите понятие *кратчайшего пути* из одной вершины ориентированного графа в другую, которое используется во многих практических приложениях. Например, взвешенным ориентированным графом моделируется транспортная сеть, по которой товары доставляются из города в город, или коммуникационная сеть, по которой переправляется информация. Рассмотрите *алгоритм Дейкстры поиска кратчайшего пути* и выполните его для заданного графа. Изучите также алгоритм *поиска дерева кратчайших путей из начальной вершины графа в любую другую его вершину*, который основан на итерационном уточнении дерева кратчайших путей, и алгоритм *поиска дерева кратчайших путей из любой вершины графа до конечной вершины*, основанный на методе динамического программирования. Выполните вручную эти алгоритмы для заданных графов и прокомментируйте полученные результаты.

Вопросы для самопроверки

1. Каковы определения понятий: граф; дуга, инцидентная вершине; вершина, коинцидентная дуге; степень вершины; граничные вершины дуги; начало и конец дуги?
2. Что понимается обычно под частичный графом, подграфом, частичным подграфом?
3. Какой смысл вкладывается в понятия взвешенной вершины, взвешенной дуги и взвешенного графа?
4. Чем отличаются ориентированные и неориентированные графы?
5. Как Вы понимаете термины: ребро; цепь; концевая вершина цепи; длина цепи; составная, сложная и простая цепь; цикл?
6. Какой граф называют связным графом?
7. Что такое компонента связности графа?
8. Сколько компонент связности может иметь связный и несвязный граф?
9. Какой смысл вкладывается в понятия: расстояние между вершинами графа; диаметр графа; путь; концевая вершина пути; начальная и конечная вершина пути; длина пути; составной, сложный и простой путь; контур?
10. Какой граф называют сильно связным графом?
11. Что такое компонента сильной связности?
12. Сколько компонент сильной связности может иметь сильно связный и несильно связный граф?
13. Каким образом представляются графы в ЭВМ с помощью матрицы смежности, матрицы инциденций и списка смежности?
14. Чем различаются возможные представления по объему занимаемой памяти и времени доступа к данным?
15. Что Вы понимаете под обходом вершин графа?
16. Каким образом выполняются обходы вершин графа в глубину и по уровням?
17. При решении каких практических задач, формулируемых в терминах графов, бывает важно определить не только число компонент связности или сильной связности графа, но и сами компоненты связности или сильной связности?
18. Как строятся матрицы достижимости, матрицы связности и матрицы сильной связности?
19. Каким образом алгоритм Уоршелла, вычисляющий матрицу достижимости, может быть использован для получения матрицы связности и матрицы сильной связности?
20. Как работает алгоритм выделения компонент связности или сильной связности, который может быть реализован на компьютере?

21. Что обычно понимают под двусвязным графом, компонентой двусвязности графа и точкой сочленения?
22. Какую практическую задачу можно решить путем анализа компонент двусвязности графа?
23. Какие способы обнаружения точек сочленения Вы знаете?
24. Как найденные точки сочленения определяют компоненты двусвязности графа?
25. Что понимают под транзитивным бинарным отношением и замыканием транзитивного отношения?
26. Какая прикладная задача может быть решена с помощью замыкания по транзитивности?
27. Каким образом замыкание транзитивного отношения выполняет известный Вам алгоритм?
28. Что обычно понимают под остовным деревом и минимальным остовным деревом?
29. В каких практических задачах может использоваться понятие минимального остовного дерева?
30. Каким образом алгоритм Дейкстры-Прима строит минимальное остовное дерево?
31. Как минимальное остовное дерево строит алгоритм Крускала?
32. Что Вы понимаете под кратчайшим путем из одной вершины ориентированного графа в другую во взвешенном и невзвешенном графах?
33. В каких практических приложениях используется понятие кратчайшего пути в графе?
34. Каким образом находит кратчайший путь в графе алгоритм Дейкстры?
35. Как можно построить дерево кратчайших путей из начальной вершины графа в любую другую его вершину с помощью алгоритма, основанного на итерационном уточнении этого дерева кратчайших путей?
36. Каким образом определяется дерево кратчайших путей из любой вершины графа до конечной вершины с использованием алгоритма, основанного на методе динамического программирования?

Литература: [4, с. 159–196], [5, с. 92–109], [6, с. 189–209], [7, с. 141–193].

Тема 5. ГЕНЕРАЦИЯ ПСЕВДОСЛУЧАЙНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ И АЛГОРИТМЫ ПОРОЖДЕНИЯ ПЕРЕСТАНОВОК

Программа

5.1. Генерация псевдослучайных последовательностей

Моделирование равномерно распределенных случайных величин. Методы моделирования дискретных и непрерывных случайных величин.

5.2. Алгоритмы порождения перестановок

Алгоритмы порождения перестановок в лексикографическом порядке, циклическим сдвигом и в порядке минимального изменения. Коды Грея.

Методические указания

5.1. Генерация псевдослучайных последовательностей

Познакомьтесь с понятием *случайных чисел* и рассмотрите примеры их использования в моделировании, выборке, численном анализе, программировании, принятии решений и защите информации. Рассмотрите понятие *закона распределения* последовательности случайных чисел и обратите внимание на то, какое распределение последовательности случайных чисел называется *равномерным*. Познакомьтесь с историей развития способов получения случайных чисел, в частности, с методом середины квадрата, предложенным Джоном фон Нейманом. Определитесь, какие последовательности чисел называются псевдослучайными. Рассмотрите методы получения последовательности случайных целых чисел, равномерно распределенных между нулем и некоторым положительным числом. Изучите *линейный конгруэнтный метод*, дающий схему наилучших датчиков случайных чисел, которые определяются четырьмя "магическими" числами: начальное значение, множитель, приращение и модуль. Обратите внимание на то, что линейные конгруэнтные последовательности всегда имеют повторяющийся цикл, называемый *периодом*. Проанализируйте, как правильно выбирать модуль, поскольку от него зависит длина периода последовательности и скорость выработки чисел. Разберитесь с тем, как правильно выбирать множитель, чтобы получать период максимальной длины. Познакомьтесь с теоремой, определяющей необходимые и достаточные условия равенства модуля и длины периода линейной конгруэнтной последовательности. Изучите понятие *мощности* линейной конгруэнтной последовательности и познакомьтесь с тем, как определяется мощность в условиях максимального периода. Рассмотрите и другие методы, предложенные для ЭВМ в качестве источников случайных чисел: квадратичный конгруэнтный метод; методы, базирующиеся на использовании для получения очередного случайного числа более одного из предыдущих значений. Рассмотрите алгоритм *A* реализации "аддитивного датчика чисел" и познакомьтесь с его анализом. Познакомьтесь с алгоритмом *M* генерации "вполне случайной последовательности" на основе двух последовательностей случайных чисел, полученных независимыми способами. Обратите особое внимание на то, что последний метод можно усиленно рекомендовать, поскольку он позволяет получать невероятно большие периоды в случаях, когда периоды исходных последовательностей являются взаимно простыми числами.

5.2. Алгоритмы порождения перестановок

Познакомьтесь с понятием *перестановки* и количеством перестановок из n элементов. Оцените возможность построения всех перестановок из n объектов в предположении, что все перестановки из $(n-1)$ объектов уже получены. Изучите метод получения всех перестановок из n натуральных чисел, в соответствии с которым для каждой перестановки из $(n-1)$ элементов образуются n других перестановок, помещением числа n на все возможные места. Рассмотрите и другой метод получения перестановок, в соответствии с которым для каждой перестановки из $\{1, 2, \ldots, n-1\}$ элементов образуются n других перестановок путем присоединения к ней элементов $1/2, 3/2, \ldots, (n-1/2)$ и дальнейшего переименования их числами $1, 2, \ldots, n$ с учетом значений и сохранением порядка элементов. Познакомьтесь с понятиями *лексикографического* и *антилексикографического* порядков. Изучите алгоритм порождения перестановок в лексикографическом порядке, который сначала генерирует перестановки в антилексикографическом порядке, а затем переразмещает их в обратном порядке.

Познакомьтесь с понятием *кода Грея* порядка n, которое определяют как любую циклическую последовательность всех наборов длины n из нулей и единиц, при этом соседние наборы отличаются ровно в одной цифре. Рассмотрите примеры использования кодов Грея при решении различных задач. В частности, познакомьтесь с использованием кодов Грея для получения правильной последовательности перемещения дисков в известной головоломке "Ханойская башня".

Вопросы для самопроверки

1. Какой смысл вкладывается в понятие случайного числа?
2. Какие примеры использования случайных чисел в моделировании, выборке, численном анализе, программировании, принятии решений и защите информации Вы знаете?
3. Что обычно понимают под законом распределения последовательности случайных чисел?
4. Какое распределение последовательности случайных чисел называют равномерным?
5. Какова история развития способов получения случайных чисел?
6. В чем заключается метод середины квадрата для получения случайных чисел, предложенный Джоном фон Нейманом?
7. Какие последовательности чисел называются псевдослучайными?
8. Какие методы получения случайных целых чисел, равномерно распределенных между нулем и некоторым положительным числом, Вы знаете?
9. В чем заключается смысл линейного конгруэнтного метода, дающего схему наилучших датчиков случайных чисел?
10. Какие четыре "магические" числа являются настроечными параметрами линейного конгруэнтного метода?
11. Что называется периодом конгруэнтной последовательности?
12. Как в линейном конгруэнтном методе правильно выбирать модуль, и какова связь значения модуля с длиной периода и скоростью выработки чисел?
13. Каким образом в линейном конгруэнтном методе необходимо выбирать множитель, чтобы получать период максимальной длины?
14. Что понимают под мощностью линейной конгруэнтной последовательности?
15. Как мощность линейной конгруэнтной последовательности определяется в условиях максимального периода?
16. Какой метод получения случайных чисел называется квадратичным конгруэнтным методом?
17. Какие методы, базирующиеся на использовании для получения очередного случайного числа более одного из предыдущих значений, Вам знакомы?
18. Каким образом алгоритм A реализует "аддитивный датчик чисел"?
19. В чем заключается главный смысл алгоритма M генерации "вполне случайной последовательности"?
20. В каком случае метод генерации "вполне случайной последовательности" позволяет получать невероятно большие периоды?
21. Что такое перестановка?
22. Каково количество перестановок из n элементов?
23. Какие методы получения всех перестановок из n объектов на основе всех перестановок из $(n-1)$ объектов Вы знаете?
24. Каким образом образуются все перестановки из $1, 2, \ldots, n$ объектов известными Вам методами?
25. Какой порядок последовательности перестановок называется лексиграфическим и какой антилексиграфическим?
26. Каким образом известный алгоритм порождает перестановки в лексиграфическом порядке?
27. Что обычно понимают под кодом Грея порядка n?
28. Каким образом выглядят коды Грея порядка 1, 2 и 3?
29. Какие примеры использования кодов Грея в решении различных задач Вам знакомы?
30. Каким образом коды Грея определяют правильную последовательность перемещения дисков в известной головоломке "Ханойская башня"?

Литература: [1, с. 75–77], [2, с. 13–51], [4, с. 283–286],
[5, с. 203–219], [6, с. 141–143].

Рекомендуемый перечень тем практических занятий

1. Понятие структуры данных. Стеки, деки, очереди, линейные списки и деревья.
2. Алгоритмы сортировки сравнениями. Вывод нижней оценки для трудоемкости работы алгоритмов данного типа.
3. Алгоритм сортировки вставками и вывод оценки его трудоемкости.
4. Обменная сортировка. Алгоритм быстрой сортировки.
5. Сортировка выбором. Турнирная и пирамидальная сортировки.
6. Сортировки слиянием и распределяющие сортировки.
7. Алгоритмы исчерпывающего поиска.
8. Алгоритмы поиска в последовательно организованных файлах.
9. Алгоритмы поиска в деревьях.
10. Хеширование и способы разрешения коллизий.
11. Способы задания графа. Остовное дерево. Алгоритм поиска в глубину.
12. Алгоритм поиска сильно связных компонент.
13. Нахождение двусвязных компонент.
14. Алгоритм построения минимального остовного дерева.
15. Моделирование равномерно распределенных случайных величин.
16. Алгоритмы порождения перестановок в лексикографическом порядке и циклическим сдвигом.

Рекомендуемый перечень тем индивидуальных занятий

1. Алгоритм сортировки вставками Шелла.
2. Обменная сортировка со слиянием алгоритм Бэтчера.
3. Лексикографическая сортировка.
4. Внешняя сортировка.
5. Алгоритмы поиска с возвращениями.
6. Индексно-последовательный поиск.
7. Построение оптимальных бинарных деревьев поиска.
8. Алгоритм сортировки в дерево.
9. Балансировка деревьев по высоте.
10. Балансировка деревьев по весу.
11. Цифровой поиск.
12. Алгоритм поиска в ширину.
13. Алгоритм нахождения кратчайшего пути.
14. Алгоритм транзитивного замыкания.
15. Алгоритм нахождения кратчайших расстояний от источника до всех остальных вершин.
16. Моделирование равномерно распределенных дискретных случайных величин.
17. Алгоритмы порождения перестановок в порядке минимального изменения.
18. Коды Грея.

Рекомендуемые примерные темы домашних заданий

1. Напишите программу, реализующую один из алгоритмов сортировки.
2. Напишите программу, реализующую один из алгоритмов поиска.
3. Напишите программу, реализующую алгоритм поиска сильно связных компонент в графе.
4. Напишите программу, реализующую алгоритм поиска двусвязных компонент в графе.
5. Напишите программу, реализующую алгоритм порождения перестановок в лексикографическом порядке.
6. Напишите программу, реализующую алгоритм порождения перестановок циклическим сдвигом.

Рекомендуемые темы лабораторных работ

1. Получить эмпирические оценки трудоемкости алгоритма сортировки или поиска.
2. Произвести сравнительный анализ трудоемкости работы нескольких алгоритмов.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Кнут, Д. Искусство программирования для ЭВМ. Т. 1. Основные алгоритмы / Д. Кнут. – М. : Мир, 1976. – 736 с.
2. Кнут, Д. Искусство программирования для ЭВМ. Т. 2. Получисленные алгоритмы / Д. Кнут. – М. : Мир, 1977. – 726 с.
3. Кнут, Д. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск / Д. Кнут. – М. : Мир, 1978. – 846 с.
4. Макконнелл, Дж. Основы современных алгоритмов / Дж. Макконнелл. – М. : Техносфера, 2004. – 368 с.
5. Нивергельт, Ю. Машинный подход к решению математических задач / Ю. Нивергельт, Дж. Фаррар, Э. Рейнголд. – М. : Мир, 1977. – 352 с.
6. Новиков, Ф.А. Дискретная математика для программистов / Ф.А. Новиков. – СПб. : Питер, 2004. – 302 с.
7. Хаггарти, Р. Дискретная математика для программистов / Р. Хаггарти. – М. : Техносфера, 2005. – 400 с.