МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Брянский государственный технический университет

Утверждаю
Ректор университета

_____О.Н. Федонин

«____»_____2020 г.

СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

АЛГОРИТМЫ БЫСТРОГО ПОИСКА

Методические указания
к выполнению лабораторной работы №10
для студентов очной, очно-заочной и заочной форм обучения
по направлениям подготовки
230100 «Информатика и вычислительная техника»,
010500 «Математическое обеспечение и администрирование информационных систем»,
231000 «Программная инженерия»

Брянск 2020

УДК 006.91

Структуры и алгоритмы обработки данных. Алгоритмы быстрого поиска [Электронный ресурс]: методические указания к выполнению лабораторной работы №10 для студентов очной, очно-заочной и заочной форм обучения по направлениям подготовки 230100 «Информатика и вычислительная техника», 010500 «Математическое обеспечение и администрирование информационных систем», 231000 «Программная инженерия». – Брянск: БГТУ, 2020.– 35 с.

Разработали:
канд. техн. наук, проф.
В.К. Гулаков
канд. техн. наук, доц.
А.О. Трубаков
канд. техн. наук, доц.
Е.О. Трубаков

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол №1 от 13.09.20)

1. ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Целью лабораторной работы является приобретение практических навыков поиска в линейных структурах на основе алгоритмов последовательного и бинарного поиска.

Продолжительность лабораторной работы − 4 часа.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Последовательный поиск

Простейшей формой поиска является последовательный поиск.

Этот поиск применяется к структурам типа массива, связанного списка или расширяющихся деревьев (splay trees).

Алгоритм поиска сводится к последовательному перебору ключей и сравнению их с заданным, зависит от вида структуры. Алгоритм поиска со вставкой также зависит от вида структуры данных. В случае использования вставки в массиве необходимо выделить достаточно памяти для этого. Он может работать в потоковом режиме при непосредственном получении данных из любого источника.

Существуют и другие приемы, повышающие эффективность вставки в массив. Например, метод транспозиции, улучшает последовательный поиск нахождением удалённой записи и вставки на её место новой записи. В случае связанного списка не надо выделять дополнительную память, более эффективно удаляются и вставляются записи. Если часто используемые записи поместить в начало файла (расширяющиеся деревья –splay trees) среднее число сравнений сильно уменьшится. Применяется для связных структур

В общем случае метод транспозиции дает более эффективный поиск, чем метод перемещения в начало списка для тех списков, в которых вероятность поиска заданного ключа остается постоянной во времени. Однако метод транспозиции требует большего времени для достижения максимальной эффективности, чем метод перемещения в начало. Другое преимущество метода транспозиции - одинаковая эффективность применения к массивам и спискам.

Рекомендуется смешанная стратегия. Вначале используется метод перемещения в начало для быстрого переупорядочивания, а затем используется метод транспозиции для поддержания списка.

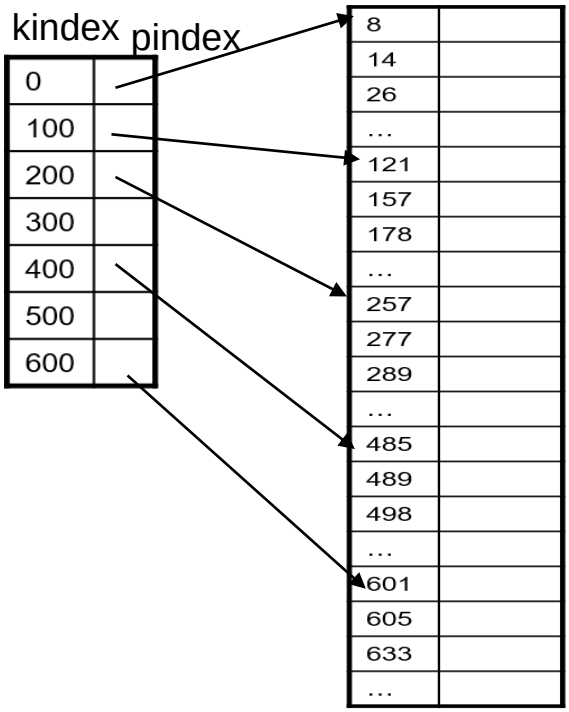
Для обнаружения записи с заданным ключом в не отсортированном списке надо n сравнений, а в отсортированном (n+1)/2 сравнений.

Если структура данных упорядочена, то существуют несколько методов увеличения эффективности поиска.

Например, если структура данных и список запросов отсортированы, то может быть выполнен последовательный поиск с одновременным продвижением и по структуре, и по списку запросов. Причем каждый последующий поиск начинается с конца предыдущего поиска.

Индексно - последовательный поиск

Суть метода - в дополнение к отсортированному файлу формируется некоторая вспомогательная таблица, называемая индексом.



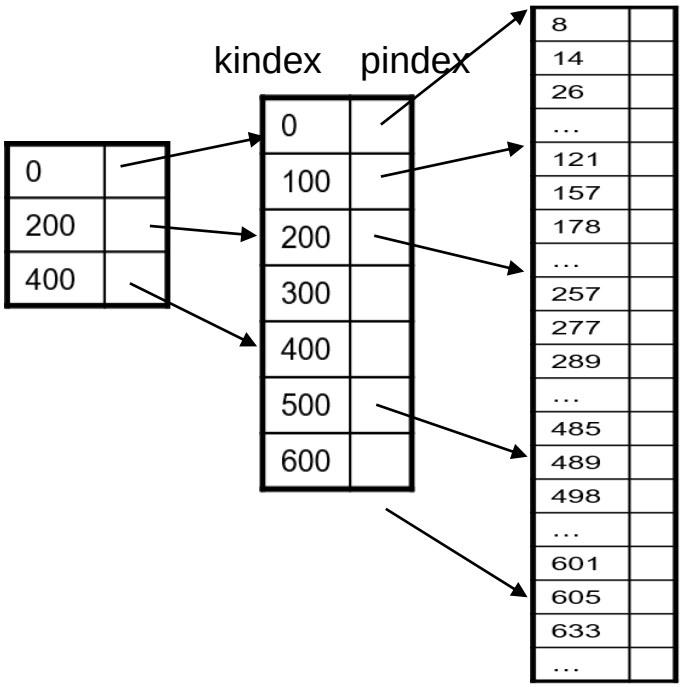
Каждый элемент индекса состоит из ключа k index и указателя на запись в файле pindex, соответствующего этому ключу. Элементы в индексе также как и элементы файла должны быть отсортированы по этому ключу

Если индекс имеет размер, составляющий одну десятую от размера файла, то каждая десятая запись представлена первоначально в индексе.

Последовательный поиск выполняется по меньшему индексу, а не по большей таблице. Когда найден правильный индекс, продолжение последовательного поиска выполняется по небольшой части записей.

Индекс применяется как для списка, так и для массива.

Использование связанного списка приводит к несколько большим накладным расходам по пространству для указателей, хотя вставки и удаления могут быть выполнены проще.



Если структура данных является такой большой, что даже использование индекса не дает достаточной эффективности, то может быть использован индекс 2 го уровня.

Индекс второго уровня действует как индекс для первичного индекса, который указывает на элементы в последовательной структуре данных.

Удаления из индексно - последовательной структуры могут быть сделаны наиболее простым способом - при помощи отметки удаленных записей флагом. Индекс изменять не надо.

Вставка в индексно - последовательную структуру является более трудной, поскольку между двумя уже существующими элементами структуры может не быть свободного места, что приводит к необходимости сдвигать большое число элементов структуры.

В общем случае, когда формируется структура данных, в ней расставляются пустые записи, чтобы оставить место для вставок

Логарифмический поиск в статических структурах

Почему логарифмический поиск? Потому что этот поиск использует древовидные структуры, а алгоритмы на основе этих структур, имеют логарифмическую сложность O(logn)

Почему в статических структурах? Потому что используемые структуры не изменяются в процессе поиска

Здесь приводятся основные методы и используемые структуры:

Бинарный поиск

Оптимальные деревья бинарного поиска

Почти оптимальные деревья бинарного поиска

Цифровой поиск
Trie-деревья
Patricia-деревья
Многопутевые Trie-деревья и другие

Наиболее эффективным методом поиска в упорядоченном массиве без использования вспомогательных индексов или таблиц является бинарный поиск *(двоичный, дихотомический)*.

При поиске в таблице суть метода состоит в том, что аргумент сравнивается с ключом среднего элемента. Если они равны, то поиск успешно закончился. В противном случае поиск должен быть осуществлен в верхней или нижней части таблицы аналогично.

Бинарный поиск наилучшим образом может быть определен рекурсивно.

Однако, большие накладные расходы, связанные с рекурсией, делают ее неподходящей для использования в практических ситуациях, в которых эффективность является главным фактором

При поиске в связной структуре аргумент поиска сравнивается с ключом, находящимся в корне. Если аргумент совпадает с ключом, поиск закончен, если же не совпадает, то в случае, когда аргумент поиска оказывается меньше ключа, поиск продолжается в левом поддереве, а в случае, когда аргумент оказывается больше ключа, в правом поддереве. Поиск считается неудачным, если при достижении листьев совпадение не обнаруживается. В противном случае к этому моменту поиск должен закончиться успехом.

Каждое сравнение в бинарном поиске уменьшает число возможных кандидатов в два раза. Следовательно максимальное число сравнений ключа равно $\log_2 n$.

Бинарный поиск может быть использован совместно с индексно-последовательной структурой данных.

К сожалению, алгоритм бинарного поиска может быть использован только для упорядоченного массива.

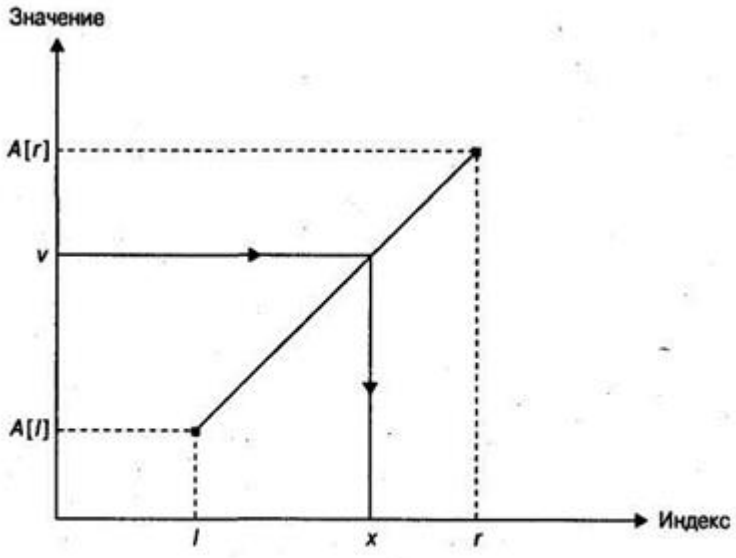
Затраты времени на поиск по бинарному дереву поиска такие же, как и затраты времени на поиск методом деления пополам.

Если данные статичны, более выгодным можно считать метод поиска делением таблицы пополам, поскольку программа при этом оказывается сравнительно простой; в представлении такой структуры данных также особой сложности нет.

Если сравнивать бинарный поиск с методом хеширования, то оказывается, что поиск по бинарному дереву хотя и уступает методу хеширования в части затрат времени на сам поиск, но имеет и свои преимущества: сравнительно высокий коэффициент использования памяти для размещения данных, возможность получать упорядоченный в алфавитном порядке список ключей.

Интерполяционный поиск

В определенном смысле алгоритм имитирует поиск фамилии в телефонном справочнике. Если нужное слово начинается с буквы 'А', вы начнете поиск где-то в начале справочника. Если Вы заметите, что искомое слово должно находиться гораздо дальше открытой страницы, вы пропустите порядочное их количество, прежде чем сделать новую попытку. Это в корне отличается от других алгоритмов, не делающих разницы между 'много больше' и 'чуть больше'.



Вычисление индекса при интерполяционном поиске

При выполнении итерации поиска между элементами A [l] (крайним слева) и A [r] (крайним справа), алгоритм предполагает, что значения в массиве растут линейно (отличие от линейности может влиять на эффективность, но не на корректность данного алгоритма).

В соответствии с этим предположением, значение v ключа поиска сравнивается с элементом, индекс которого вычисляется (с округлением) как координата x точки на прямой, проходящей через (l, A [l]) и (r, A [r]), координата y которой равна значению v (см. рис).

Логика, лежащая в основе этого метода, проста. Мы знаем, что значения массива возрастают (точнее говоря, не убывают) от A [l] до A [r], но не знаем, как именно. Пусть это возрастание — линейное (простейшая из возможных функциональных зависимостей); в таком

случае вычисленное по формуле значение индекса — ожидаемая позиция элемента со значением, равным v.

Записав стандартное уравнение для прямой, проходящей через две точки (l, A [l]) и (r, A [r]), заменив в нем y на v и решая его относительно x, получим формулу.

$$x = l + \left\lfloor \frac{(v - A[l])(r - l)}{A[r] - A[l]} \right\rfloor$$

После сравнения v с A[x] алгоритм либо прекращает работу (если они равны), либо продолжает поиск тем же способом среди элементов с индексами либо от l до x -1, либо от x + 1 до r, в зависимости от того, меньше ли v значения A [x] или больше.

Пример. Если взять последовательность с линейным возрастанием

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,

то для поиска заданного ключа методом бинарного поиска необходимо четыре сравнения, а методом интерполяционного поиска одно сравнение.

Если взять случайно возрастающую последовательность

9, 17, 25, 33,34, 35, 49, 67, 69, 85, 86, 94, 96, 105, 106, 108, то для поиска заданного ключа (например X=33) методом бинарного поиска требуется 4 сравнения, а методом интерполяционного поиска 3 сравнения.

Анализ показывает, что интерполяционный поиск в среднем требует менее $\log_2 \log_2 n+1$ сравнений ключей при поиске в списке из n случайных значений.

Эта функция растет настолько медленно, что для всех реальных практических значений n ее можно считать константой.

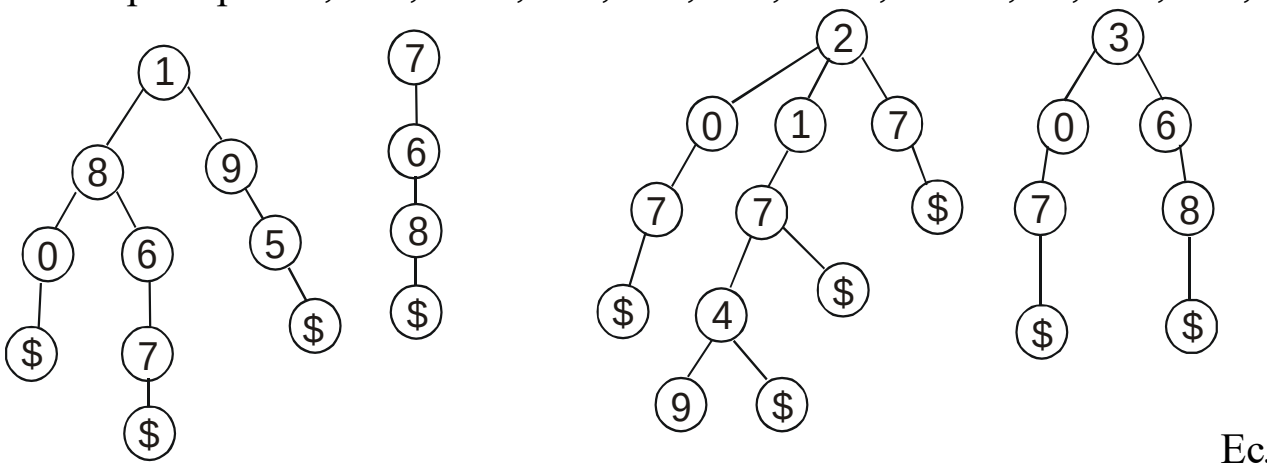
Однако в наихудшем случае интерполяционный поиск вырождается в последовательный, который рассматривается как наихудший из возможных (почему?).

При сравнении интерполяционного поиска с бинарным приведем мнение Седжвика, считающего, что бинарный поиск, вероятно, более выгоден для небольших входных данных, но для файлов большого размера и для приложений, в которых сравнение или обращение к данным — дорогостоящая операция, лучше использовать интерполяционный поиск.

Деревья цифрового поиска (digital search trees - DST) представляют собой n-арные деревья, ветвление в которых

выполняется не в соответствии с результатом сравнения полных ключей, а в соответствии с выбранными разрядами ключа.

Пример. 180, 195, 1867, 768, 207, 217, 2174, 21749, 27, 307, 368;



Есл

и ключи состоят из символов алфавита, то каждая буква определяет некоторую ветвь дерева. Специальный символ в конце каждого листа должен содержать указатель на запись

Производительность для худшего случая деревьев, построенных по методу поразрядного поиска, значительно выше производительности для худшего случая бинарных деревьев, при условии - если количество ключей велико, а длина ключей мала по сравнению с их количеством.

Количество сравнений никогда не превышает количество разрядов в ключе поиска.

Большие DST-деревья, образованные случайными ключами, часто почти идеально сбалансированы.

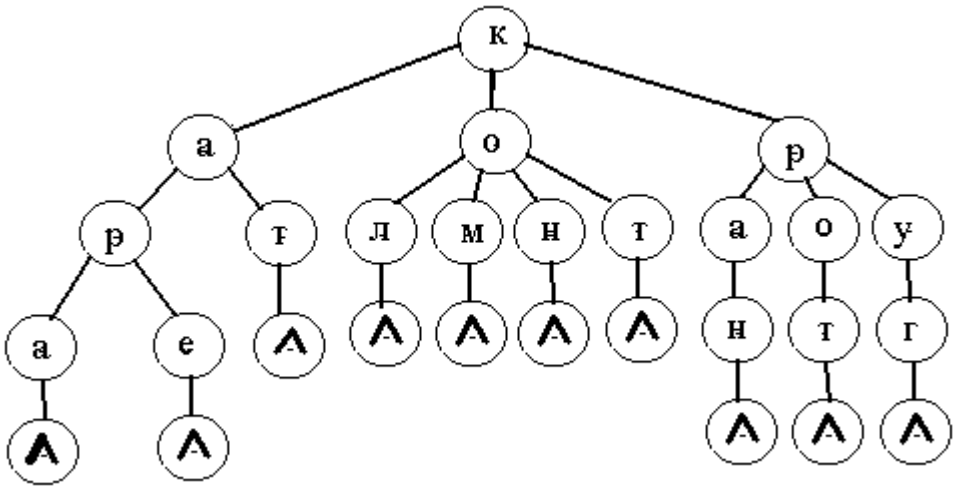
Алгоритм поиска и вставки с анализом приведен у Кнута

Можно организовать поиск только по небольшому списку сыновей для того, чтобы узнать, появляется ли заданный символ в некоторой заданной позиции ключей.

Trie-деревья (нагруженные деревья)

Обычно в узлах дерева поиска хранятся значения ключей, но в случае, когда ключами являются достаточно короткие слова, можно рассматривать каждый ключ как список букв, а все списки вместе— как дерево поиска, структура которого несколько отличается от рассмотренных ранее.

В этой структуре узлу (i+1)-го уровня ставится в соответствие i-я буква слова, так что каждый узел содержит только один символ. Методы поиска по такому дереву часто весьма экономичны как по памяти, так и по времени



trie-деревья представляют собой структуры данных, применение которых не уступает по эффективности методам хеширования.

К достоинствам нагруженных деревьев можно отнести возможность перемещения по дереву и выполнения различных операторов за время, пропорциональное длине «обслуживаемого» слова.

Другим достоинством нагруженных деревьев является то, что, в отличие от хеш-таблиц, они поддерживают эффективное выполнение оператора MIN

Хеш-функция, чтобы быть действительно «случайной», хеширует каждый символ слова. И, конечно, время вычисления хеш-функции не включает время, необходимое для разрешения коллизий или выполнения операций вставки, удаления или поиска.

Поэтому мы вправе ожидать, что нагруженные деревья будут работать значительно быстрее со словарями, состоящими из символьных строк, чем хеш-таблицы.

Основанный на trie-деревьях поиск обладает двумя недостатками: однонаправленное ветвление приводит к созданию дополнительных узлов в trie-дереве, что кажется необязательным; в trie-дереве присутствуют два различных типа узлов, что приводит к усложнениям.

В 1968 г. Моррисон (Morrison) изобрел способ ликвидации обоих проблем путем применения метода, который назвал patricia (practical algorithm to retrieve information coded in alphanumeric - практический алгоритм получения информации, закодированной алфавитно - цифровыми символами).

Точный анализ среднего случая patricia-дерева сложен; из него следует, что в среднем в patricia-дереве требуется на одно сравнение меньше, чем в стандартном trie-дереве

Рассмотренные методы целесообразно применять к мало изменяющимся данным. В противном случае эти структуры могут выродится и поиск превратится в последовательный

Логарифмический поиск в динамических таблицах

Случайные деревья бинарного поиска

AVL-деревья

BB-деревья

RB-деревья

B-деревья

В случае, когда множество ключей заранее неизвестно или когда это множество ключей меняется, вставки и удаления ключей в таблице оказываются довольно трудоемкими. Более рационально использовать в таком случае бинарное дерево поиска, которое позволяет значительно проще вставлять и удалять элементы.

Например, если необходимо построить таблицу частоты использования отдельных слов в некотором тексте на естественном языке, то для представления таблицы в памяти лучше использовать бинарное дерево поиска.

Существуют алгоритмы, которые позволяют построить оптимальное дерево поиска. К ним относится, например, алгоритм Гарсия–Воча. Однако такие алгоритмы имеют временную сложность порядка $O(n^2)$. Таким образом, создание оптимальных деревьев поиска требует больших накладных затрат, что не всегда эффективно при быстром поиске.

3. ТЕХНИКА БЕЗОПАСНОСТИ ПРИ ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Меры безопасности при работе с электротехническими устройствами соответствуют мерам безопасности, принимаемым при эксплуатации установок с напряжением до 1000 В и разработанным в соответствии с «Правилами техники безопасности при эксплуатации электроустановок потребителей», утвержденными Главгосэнерго-надзором 21 декабря 1984 г.

Студенту не разрешается приступать к выполнению лабораторной работы, если замечены какие-либо неисправности в лабораторном оборудовании.

Студент не должен прикасаться к токоведущим элементам электрооборудования, освещения и электропроводке, открывать двери электрошкафов и корпусов системных блоков, мониторов.

Студенту запрещается прикасаться к неизолированным или поврежденным проводам и электрическим устройствам, наступать на переносные электрические провода, лежащие на полу, самостоятельно ремонтировать электрооборудование и инструмент.

Обо всех замеченных неисправностях электрооборудования студент должен немедленно поставить в известность преподавателя или лаборанта.

При выполнении лабораторной работы необходимо соблюдать осторожность и помнить, что только человек, относящийся серьезно к своей безопасности, может быть застрахован от несчастного случая.

4. УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ

Каждую задачу лабораторной работы решить в соответствии с изученным алгоритмами последовательного и бинарного поиска в линейных структурах, реализовав программный код на языке C++. Программу для решения каждой задачи разработать методом процедурной абстракции, используя рекурсивные функции. Этапы сопроводить комментариями в коде. В отчете о лабораторной работе отразить разработку и обоснование математической модели решения задачи. Результаты тестирования программ провести в соответствии приведенными примерами входных и выходных файлов к задачам (как дополнение допустимы и собственные примеры тестовых данных).

Каждую задачу реализовать в соответствии с приведенными этапами:

- изучить словесную постановку задачи, выделив при этом все виды данных;
- сформулировать математическую постановку задачи;
- выбрать метод решения задачи, если это необходимо;
- разработать графическую схему алгоритма;
- записать разработанный алгоритм на языке C++;

- разработать контрольный тест к программе;
- отладить программу;
- составить отчет о лабораторной работе.

5. ЗАДАЧИ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Создать индексный файл для файла записей отсортированный по некоторому ключевому полю, запоминая индекс каждой М-ой записи. Организовать главный индекс в оперативной памяти, запоминая доступ к каждой М-ой записи индексного файла. Обеспечить выдачу записи с заданным ключевым путем последовательного поиска в главном индексе и бинарного поиска в индексном файле.

2. В системе бронирования авиационных билетов имеется следующая информация:

1) перечень рейсов;

2) список пассажиров на каждый рейс;

3) очередь пассажиров в случае, когда число заявок на билеты превышает количество мест. при бронировании имя пассажира вносится в список соответствующего рейса либо включается в очередь на данный рейс. Организовать в основной памяти указанную информацию. Перед включением в любой список провести проверку на отсутствие пассажира в данном списке. Обеспечить бинарный поиск нужного рейса и хеширование очередей пассажиров при выполнении запроса на бронирование. Ограничить величину каждой очереди количеством мест на соответствующем рейсе.

3. Имеется файл фамилий в алфавитном порядке. Создать индексный файл для поиска фамилий. Задается список лиц, изменивших фамилии. Провести корректировку исходного и индексного файлов.

4. Вводится файл о неуспевающих студентах со следующей структурой записей:

Факультет

Группа

Фамилия студента
Число хвостов

Записи по каждому факультету, а также по каждой группе следуют друг за другом. После каждого экзамена число хвостов у студентов увеличивается. Требуется обеспечить коррекцию информации с помощью индексирования файла. Для расширения файла необходимо зарезервировать файл переполнения. Организовать заставку путем помещения записи в файл переполнения и связи ее с соседними записями. Предусмотреть автоматическую реорганизацию файла неуспевающих, если число записей переполнения превышает заданное число N.

5. Имеется файл записей с некоторым ключевым полем. Отсортировать файл методом простого слияния. Обеспечить поиск записи по ключу методом бинарного поиска.

6. Имеется файл записей с некоторым ключевым полем. Отсортировать файл пирамидальной сортировкой. Обеспечить поиск записи по ключу двумя методами поиска.

7. Записи файла имеют следующую структуру:
Факультет
Группа
Фамилия студента
Средний балл

Записи по каждому факультету, а также в пределах каждой группы расположены в файле друг за другом. Организовать двухуровневый индекс и с его помощью определить средний балл указанной группы.

8. В файле записаны фамилии студентов и их анкетные данные, включающие номера паспорта и зачетной книжки. Организовать два индексных файла для быстрого поиска информации по любому из этих номеров.

9. Записи файла о неуспевающих студентах имеют следующую структуру:
Факультет

Группа
Фамилия студента
Число хвостов

Файл частично отсортирован. Записи по каждому факультету, а также по каждой группе расположены в файле друг за другом. После пересдачи экзаменов число хвостов у студентов уменьшается. Требуется обеспечить коррекцию информации с помощью индексирования. Если студент выбывает из должников, пометить соответствующую запись специальным флагом. Написать программу копирования файла с физическим удалением помеченных записей.

10. Вводится файл о неуспевающих студентах со следующей структурой записей:

Факультет
Группа
Фамилия студента
Число хвостов

Записи по каждому факультету, а также по каждой группе следуют друг за другом. После каждого экзамена число хвостов у студентов увеличивается. Факультеты на основании многолетней статистики заранее сообщают о максимально возможном количестве должников. Требуется обеспечить коррекцию информации с помощью индексирования файла. Для расширения файла необходимо зарезервировать области переполнения по каждому факультету. Организовать вставку путем сдвига записей вперед до ближайшей области переполнения. Предусмотреть реорганизацию файла неуспевающих в окончательной редакции.

11. Реализуйте алгоритмы последовательного поиска и последовательного поиска и вставки для массивов и связанных списков.

12. Имеется n монет, среди которых одна фальшивая (легче или тяжелее других монет). Дополнительно дается одна настоящая. Имеются весы, с помощью которых можно сравнить различные группы монет. Разработать алгоритм и программу, позволяющую при

минимальном числе сравнений определить фальшивую монету (если она есть) .

13. Задан большой массив записей, хранящийся на диске (массив не помещается полностью в оперативную память). В каждой записи имеется как минимум два поля A и B. Необходимо:

1) создать два массива индексов 1 и 2, отсортированных по возрастанию значений полей A и B соответственно;

2) произвести выборку полей для заданных диапазонов изменения поля A и B (должны выполняться два условия одновременно).

14. В большом файле (более 1000 записей) необходимо найти 100 ближайших к заданному ключу записей, т.е. Записей для которых функция расстояния $d(k,k_j)$ принимает наименьшие значения. Выбрать подходящую структуру данных для такого последовательного поиска и написать соответствующую программу.

15. Разработать программу, реализующую гибридный метод поиска, позволяющий переходить от бинарного поиска к последовательному при достаточном уменьшении интервала поиска. Определить наилучший момент перемены метода поиска.

16.Написать эффективную программу, позволяющую найти 10% записей с максимальными ключами

17.Написать программу интерполяционного поиска в таблице символов размером n. Экспериментально найти значения n, при которых интерполяционный поиск в 1, 2, 10 раз быстрее бинарного поиска.

18. Имеется файл с наименованиями компьютерных комплектующих в алфавитном порядке. Создать индексный файл для поиска наименования товара.

19. Имеется файл с наименованиями компьютерных комплектующих в алфавитном порядке. Реализуйте метод деления пополам и интерполяционный поиск. Сравните быстродействие подходов. Реализовывать индексный файл не нужно.

20. Файл с фамилиями из паспортного стола отсортирован в алфавитном порядке. Создать индексный файл для поиска информации по фамилии.

21. Файл с фамилиями из паспортного стола отсортирован в алфавитном порядке. Реализуйте метод деления пополам и интерполяционный поиск. Сравните быстродействие подходов. Реализовывать индексный файл не нужно.

22. Файл с учетками банковской системы отсортирован в алфавитном порядке по номеру карты. Создать индексный файл для поиска информации о клиенте по номеру карты.

23. Файл с учетками банковской системы отсортирован в алфавитном порядке по номеру карты. Реализуйте метод деления пополам и интерполяционный поиск. Сравните быстродействие подходов. Реализовывать индексный файл не нужно.

24. Файл с учетками системы доставки еды отсортирован в алфавитном порядке по фамилии клиента. Создать индексный файл для поиска информации о клиенте по фамилии.

25. Файл с учетками системы доставки еды отсортирован в алфавитном порядке по фамилии клиента. Реализуйте метод деления пополам и интерполяционный поиск. Сравните быстродействие подходов. Реализовывать индексный файл не нужно.

26. Файл с названием городов мира отсортирован в алфавитном порядке. Создать индексный файл для поиска информации о городе по названию города.

27. Файл с названием городов мира отсортирован в алфавитном порядке. Реализуйте метод деления пополам и интерполяционный поиск. Сравните быстродействие подходов. Реализовывать индексный файл не нужно.

28. Файл с базой автомобилей отсортирован в алфавитном порядке. Создать индексный файл для поиска владельца по номеру автомобиля.

29. Файл с базой автомобилей отсортирован в алфавитном порядке. Реализуйте метод деления пополам и интерполяционный поиск. Сравните быстродействие подходов. Реализовывать индексный файл не нужно.

30. Файл с информацией о студентах отсортирован в алфавитном порядке по номеру зачетки. Создать индексный файл для поиска ФИО студента по номеру зачетки.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Когда применяется последовательный поиск? Какова его сложность?
2. В чем заключается особенность индексно-последовательного поиска? Какова его сложность?
3. В чем заключается особенность интерполяционного поиска?
4. Какие виды поиска Вы знаете в рамках логарифмического (быстрого) поиска в статических таблицах?
5. Какие виды поиска Вы знаете в рамках логарифмического (быстрого) поиска в динамических таблицах?
6. Чем можно объяснить многообразие алгоритмов поиска в линейных структурах?
7. В чем заключаются преимущества поиска с барьером по сравнению с последовательным поиском?
8. Нахождение какого по порядку элемента в линейном множестве (первого, последнего) гарантирует алгоритм прямого поиска? Как в этом случае должен быть выполнен просмотр?
9. Нахождение какого по порядку элемента в линейном множестве (первого, последнего) гарантирует алгоритм бинарного поиска? Ответ обоснуйте.
10. Как трудоемкость алгоритма бинарного поиска на дискретном множестве зависит от мощности множества?

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. 3. Кнут, Д. Искусство программирования для ЭВМ Т. 3. Сортировка и поиск / Д. Кнут – М: Вильямс, 2000. – 245 с.

2. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт – М.: ДМК Пресс, 2010 – 272 с.: ил. ISBN 978 5 94074 584 6

3. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М: Издательский дом «Вильямс», 2005. –1296 с. ISBN 5-8459-0857-4(рус)

4. Ахо, А. В. Структуры данных и алгоритмы / А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман. – М.: Вильямс, 2003. – 382 с. ISBN 10:0-201-00023-7

5. Седжвик, Р. Фундаментальные алгоритмы на C++. Анализ. Структуры данных. Сортировка. Поиск: пер. с англ. /Р. Седжвик. – К.: ДиаСофт, 2001. – 688 с. ISBN 5-93772-054-7

6. Хэзфилд, Р. Искусство программирования на С. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста: пер. с англ. / Р. Хэзфилд, Л. Кирби [и др.]. – К.: ДиаСофт, 2001. – 736 с. ISBN 966-7393-82-8, 0-672-31896-2

7. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си (+ CD): учеб. Пособие / Б.С. Хусаинов. – М.: Финансы и статистика, 2004. – 464 с. ISBN: 5-279-02775-8

19. Гулаков В. К. Введение в хеширование: монография/ В.К. Гулаков, К. В. Гулаков – Брянск: БГТУ, 2011. – 129 с. ISBN 5-89838-593-4.

19. Гулаков, В.К. Деревья: алгоритмы и программы / В.К. Гулаков. – М.: Машиностроение-1, 2005. – 206 с.

20. Гулаков, В.К. Многомерные структуры данных / В.К. Гулаков, А.О. Трубаков. – Брянск: БГТУ, 2010. – 387 с.

Структуры и алгоритмы обработки данных. Алгоритмы быстрого поиска: методические указания к выполнению лабораторной работы №10 для студентов очной, очно-заочной и заочной форм обучения по направлениям подготовки: 230100 «Информатика и вычислительная техника», 010500 «Математическое обеспечение и администрирование информационных систем», 231000 «Программная инженерия»

ВАСИЛИЙ КОНСТАНТИНОВИЧ ГУЛАКОВ
АНДРЕЙ ОЛЕГОВИЧ ТРУБАКОВ
ЕВГЕНИЙ ОЛЕГОВИЧ ТРУБАКОВ

Научный редактор В.В. Конкин
Редактор издательства Л.Н. Мажугина
Компьютерный набор В.К. Гулаков

Темплан 2020 г., п. 206

Подписано в печать Формат 1/16
Усл.печ.л. 2,03 Уч.-изд.л. 2,03

Издательство Брянского государственного технического университета
241035, Брянск, бульвар им.50-летия Октября, 7, БГТУ, тел. 58-82-49
Лаборатория оперативной полиграфии БГТУ, ул. Институтская, 16