Часть 1 (Логика)

Логические операции

Исключающее ИЛИ

Этот логический оператор отстутствует в ряде ЯПВУ.
Его можно имитировать следующим образом.

```
A ? (A && !B) : (!A && B);
```

это код на языке `C`, задействующий тернарный оператор, и логические высказывания в его составляющих. Здесь `A`, `B` – операнды имитируемой операции XOR.
Работа этого выражения описывается следующей таблицей истинности.

```
A       B       Result
0       0       0
0       1       1
1       0       1
1       1       0
```

Как видно из таблицы истинности, это выражение имитирует операцию XOR.

Импликация

Эта операция обозначается как A → B, и трактуется как «ложь — если из истины следует ложь, иначе — истина».
Этот логический оператор отстутствует в ряде ЯПВУ.
Его можно имитировать следующим образом.

```
!A || B;
```

это код на языке `C`, задействующий логические операторы. Здесь `A`, `B` – операнды имитируемой операции →.
Работа этого выражения описывается следующей таблицей истинности.

```
A       B       Result
0       0       1
0       1       1
1       0       0
1       1       1
```

Как видно из таблицы истинности, это выражение имитирует операцию →.

Часть 2 (Сортировка)

Сортировка и поиск

Сортировка и поиск — классические задачи информатики.
Алгоритмы сортировки и поиска изложены в следующих подглавах.

Сортировка

Сортировка — это собственно сортировка данных по целевому критерию.
В целях простоты изложения будет рассмотрена сортировка массивов.

Сортировка выбором

Алгоритм:

1. находим наименьший элемент массива; перемещаем его в начало массива;
2. находим наименьший элемент оставшейся части массива; перемещаем его в начало оставшейся части массива;
3. повторяем предыдущий шаг итеративно, до перебора предпоследнего элемента массива (после перебора предпоследнего элемента массива сортировка будет завершена — действительно, после этого останется лишь один последний элемент, а сортировать один единственный элемент не имеет смысла);

Коментарий:
найти наименьший элемент можно следующим простейшим способом — последовательно сравнивать элементы массива с его первым элементом, и условно обменивать его с ним; тогда, в результате, первым окажется наименьший элемент массива (действительно, каким бы ни был первый элемент массива изначально, в ходе последовательных сравнений (и условных обменов) на его месте окажется самый меньший элемент);

Вычислительная сложность:
алгоритм делает n машинных циклов при первом проходе, и n − m на последующих (m − номер прохода), итого — (в интересах ассимптотической оценки будем считать n − m = n) n проходов по n элементов в каждом проходе, что результирует значением n^2;
за проход считать сравнение элементов, и их условный обмен;

Сортировка вставкой

Алгоритм:

1. сравниваем первую пару элементов, и условно их меняем местами; при этом, всплывший элемент сравниваем с последующим, и так итеративно, до конца массива;
2. смещаем фокус внимания на один элемент; повторяем шаг один;
3. повторяем предыдущий шаг итеративно, до смещения фокуса на предпоследний элемент (тогда будет пройден весь массив — действительно, ведь сравнение происходит по парам элементов);

Комментарий:
этот алгоритм работает так — после шага один на верху оказывается элемент (имеющий значение- экстремум), последующий шаг сортирует оставшуюся часть массива, и, таким образом, сортировка со всплывшим элементом не требуется; это касается и всех последующих шагов;
необходимо помнить об этой особенности алгоритма, дабы эффективно его реализовать;
это и есть пресловутая «пузырьковая сортировка»;

Вычислительная сложность:
алгоритм делает n машинных циклов при первом проходе, и n – m на последующих (m – номер прохода), итого — (в интересах ассимптотической оценки будем считать n – m = n) n проходов по n элементов в каждом проходе, что результирует значением n^2;
за проход считать сравнение элементов, и их условный обмен;

Быстрая сортировка

Алгоритм:

1. выполнить сортировку элементов относительно середины массива;
2. повторить шаг один для частей массива (располагающихся по краям середины), и так итеративно, покуда части массива (располагающихся по краям середины) можно делить посередине;

Комментарий:
при итеративном дроблении массива по середине, в результате все элементы массива будут отсортированы (действительно, если дробление произведено итеративно, до предела дробимости, то приходится говорить о том, что элементы отсортированы не только относительно нее, но и относительно друг друга вообще);
в интересах реализации этого алгоритма необходимо помнить следующее: если при вычислении индекса середины массива будет получено нецелое значение — то его следует округлить, если дробимая часть массива окажется вырожденной (насчитывающей всего два элемента) — то вместо ее дробления следует перейти к сравнению этих двух элементов;

это один из самых быстрых алгоритмов сортировки;

Вычислительная сложность:
алгоритм делает n машинных циклов при первом проходе, и 2*n/2 на последующих, итого — n проходов по n элементов в каждом проходе, что результирует значением n^2; однако, на практике, этот алгоритм не приводит к необходимости перемещать все элементы на каждом шаге (перемещение — наиболее затратная операция, а частичная сортировка практически нивелирует эту необходимость); таким образом, необходимость перемещения уменьшается с каждым проходом, и обратно пропорциональна номеру прохода (с каждым проходом снижается вдвое); на практике алгоритм следует оценивать так — n проходов по 2/n перемещений в каждом, что результирует значением n log(n), соответственно O(n) = log(n).
за проход считать сравнение элементов, и их условный обмен;

Поиск

Поиск — это собственно поиск целевого значения. Для поиска отсортированных и неотсортированных массивах служат соответствующие алгоритмы.
В целях простоты изложения поиск будет рассматриваться на массивах.

Линейный поиск

Алгоритм:

1. проверить элемент массива на целевое значение, при совпадении с ним вернуть этот элемент;
2. повторить шаг один со следующим элементом, и так итеративно, до конца массива;

Комментарий:
этот алгоритм применяется только к небольшим массивам (по причине его тривиальной логики и соответственно низкой производительности);
элементарный алгоритм линейного поиска наименьшего элемента был рассмотрен выше, в пг. Сортировка п. Сортировка выбором — на первом шаге этого алгоритма будет найден наименьший элемент, и в результате этого шага он будет первым элементом массива; это простейший алгоритм, позволяющий найти наименьший элемент в неотсортированном массиве;

Вычислительная сложность:
всего алгоритм делает n шагов — итого — один шаг по n элемнтов в нем, что результирует зачением n;

Направленный поиск

Направленный поиск применяется когда заведомо целесообразно то или иное направление обхода при поиске. Определенное направление поиска заведомо целесообразно при поиске по иерархическим структурам данных.

Поиск в глубину

Алгоритм:

1. спуститься по верхней размерности массива до последней его размерности, перебрать элементы этой размерности (на предмет наличия целевого значения);
2. повторить шаг один со следующей верхней размерностью массива, и так итеративно, по всем размерностям;

Комментарий:
как следует из названия, поиск производится вглубь иерархической структуры данных; ради простоты, этот алгоритм рассматривается на многомерных массивах; он может использоваться также на деревьях, тогда он должен учитывать организацию дерева (в отличии от массива, дерево может быть иррегулярной структурой);

Поиск в ширину

Алгоритм:

1. перебрать верхнюю размерность массива, переключиться на следующую размерность того же уровня иерархии, и так итеративно, до конца всех размерностей этого уровня;
2. повторить шаг один для размерностей следующего уровня иерархии, и так итеративно, до конца всех уровней иерархии;

Комментарий:
как следует из названия, поиск производится вширь иерархической структуры данных; ради простоты, этот алгоритм рассматривается на многомерных массивах; он может использоваться также на деревьях, тогда он должен учитывать организацию дерева (в отличии от массива, дерево может быть иррегулярной структурой);

Вычслительная сложность направленного поиска

Вычислительная сложность:
алгоритм в худшем случае перебирает все элементы, всего алгоритм делает n шагов — итого — один шаг по n элемнтов в нем, что результирует зачением n; однако, на практике, этот алгоритм не приводит к необходимости искать элементы на каждом шаге (предполагается, что при направленном поиске

вероятность обнаружения целевого элемента не равна на протяжении всего массива); таким образом, вероятность нахождения увеличивается с каждым проходом по следующей размерности массива, и прямо пропорциональна номеру прохода (с каждым проходом повышается); на практике алгоритм следует оценивать так — алгоритм перебирает m элементов на первом шаге (он перебирает элементы его первой размерности), и 2/x*m на последующих шагах (x – номер перебираемой последовательности, его необходимо учесть, так как при направленном поиске предполагается — вероятность нахождения в каждой следующей размерности выше(иначе зачем производить направленный поиск?)), соответственно вычислительная сложность вычисляется как у алгоритма быстрой сортировки, O(n) = log(n), если считать m = n.

Бинарный поиск

Алгоритм:

1. сравнить середину массива с целевым значением, если целевое значение обнаружено, то вернуть этот элемент;
2. повторить шаг один, применив его к половине массива, и так итеративно, до нахождения элемента с целевым значением;

Комментарий:
этот алгоритм целесообразно применять к предварительно отсортированным массивам, только тогда его эффективность оказывается высока (действительно, только тогда возможно рационально выбрать направление дробления массива пополам, и тем самым ускорить поиск); иначе — его эффективность сравнима с эффективностью линейного поиска;

Вычислительная сложность:
вычислительная сложность вычисляется как у направленного поиска (так как вероятность обнаружения элемента возрастает скаждым шагом алгоритма), O(n) = log(n);

Тернарный поиск

Алгоритм:

1. разделить поле поиска на трети, и искать в средней трети;
2. искать в остальных третях;

Комментарий:
этот алгорит целесообразно применять к предварительно отсортированным массивам (причем, отсортированным специфическим образом — при котором порядок сортировки меняется от начала к концу массива);

Вычислительная сложность:

вычислительная сложность вычисляется как у направленного поиска (так как вероятность обнаружения элемента возрастает скаждым шагом алгоритма), O(n) = log(n);

Резюме

Алгоритмы сортировки в худшем случае делают n проходов по n элементов; соотетственно, их вычислительная сложность равна n^2.
Алгоритмы поиска в худшем случае делают поиск по всем элементам; соответственно, их вычислительная сложность равна n.

Часть 3 (Хэш)

Хэширование

Хэширование - «свертка», преобразование массива, при котором результатом преобразования является значение, размер которого равен размеру элемента массива. При этом, не должно быть очевидного соответствия между входными данными и результатом (по результату должно быть сложно восстановить входные данные). И, при этом, разные входные данные должны приводить к формированию разных значений результата. Возникновение одинаковых значений результата называется «коллизией».
Существуют различные алгоритмы хэширования.

Алгоритм CRC

CRC – Cyclic Redundancy Code.
Алгоритм работает так. Входной массив трактуется как длинное число. Это число делится на операнд размера элемента массива. Результирующий остаток от деления и есть результат алгоритма.
В интересах вычислительного процесса деление имитируется, оно заменяется секвенцией вычитаний (вычитаемого из уменьшаемого). Это имеет смысл, так как схемы (выполняющие алгебраическую сумму) работают быстрее, чем схемы (выполняющие умножение). В интересах оптимизации циклического вычитания оно заменяется на вычитание по модулю 2; модульная арифметика (по модулю 2) специфична тем, что все операции производятся по модулю 2, и перенос (из разряда в разряд) отсутствует. Вычитание по модулю 2 осуществляется операцией XOR. Это возможно благодаря отсутствию переноса в этой арифметике.
Что касается используемого в алгоритме делителя — то он представляет собой определенное значение, подобранное в интересах использования алгоритма.
Существуют различные реализации алгоритма (они оперируют с делителем разной разрядности). Алгоритм CRC-8 оперирует с делителем размером 1 byte (значение делителя $x^8 + x^7 + x^6 + x^4 + x^2 + 1$, x – установленные биты), CRC-16 – 2 byte (значение делителя $x^{16} + x^{15} + x^2 + 1$, x – установленные биты), CRC-32 – 4 byte (значение делителя $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$, x – установленные биты).
При реализации вышеописанного алгоритма необходимо помнить, что этот алгоритм применим, только если входной массив достаточно мал, чтобы остаток от деления уместился в операнде, равном по размеру делителю. Этот алгоритм сохраняет остаток от деления в операнде размером с делитель. Таким образом, максимально возможное количество уникальных хэшей определяется размером делителя (так как ему равен размер остатка от деления). Так как хэш должен однозначно идентифицировать входной массив, то максимальное число

различных входных массивов определяется максимальным количеством уникальных хэшей. Максимальный объем входного массива определяется тем, уложится ли остаток от его деления в размер делителя (которому равен размер остатка от деления). Максимальный размер остатка от деления равен размеру делителя (так как если же его размер больше делителя, то деление возможно). Таким образом, максимальный размер входного массива может быть любым, так как максимальный размер остатка от деления (не важно, сколь большого делимого) всегда равен размеру делителя (согласно вышеописанному).
Также необходимо помнить, что в исторической перспективе развития алгоритма детали его реализации могут меняться. В особенности — значения используемых делителей.

О прочих алгоритмах хэширования

Прочие практически используемые алгоритмы хэширования не тривиальны, и их понимание требует профессиональной математической подготовки.

Часть 4 (Об алгоритмах)

Об алгоритмах

Алгоритм — это собственно алгоритм решения определенной задачи.
Время работы алгоритма (в зависимости от объема входных данных) принято обозначать T(n). Время работы алгоритма принято оценивать ассимптотически — предельно, так как при больших объемах входных данных сам порядок роста функции важнее фигурирующих в ней коэффициентов и прочих значений. Ассимптотическую оценку времени работы алгоритма принято обозначать O(n).
Время работы алгоритма также может зависеть от входных данных (не их объема, а самих входных данных — их значений и их последовательности), если конечно сама логика работы алгоритма предусматривает эту зависимость.
Ассимптотическую оценку наименьшего времени работы алгоритма принято обозначать W(n) - "Омега большое от n". Если наименьшая и наибольшая (O(n), и W(n)) ассимптотические оценки равны, то оценку принято обозначать Theta(n) - "Тэта большое от n". Разумеется, эти оценки имеют смысл, только когда логика работы алгоритма зависит от входных данных.