Одномерные массивы. Алгоритмы работы с массивами

Массивы

Массивы предназначены для хранения наборов данных. Хранятся в памяти последовательно, что делает возможным одномерную индексацию (произвольный доступ к элементу массива). Синтаксис:

```
тип имя[размер];
```

где размер — константа (число или именованная константа). Элементы массива нумеруются с нуля.

Пример:

```
const int n=5;
int mas[n];
```

Массив можно инициализировать при объявлении:

```
int mas[n]={3, -4, 1, 0, -7};
```

Если число инициализаторов меньше размера массива, то остальные элементы заполнятся нулями. Для обработки массивов используется цикл for.

Работа с функциями

Массив передается в функцию по адресу. При передаче в функцию информация о размере массива теряется, поэтому необходимо передавать в нее не только сам массив, но и его размер.

Пример: см. задачу 1 https://replit.com/@IekatierinaLat1/Arrays#main.cpp

Динамические массивы

Размер динамического массива задается во время выполнения программы. Для работы с динамическим массивом необходимо выделить память. Синтаксис:

```
тип *имя = new тип[размер];
```

где размер — переменная, которой до объявления массива должно быть присвоено значение (положительное целое число).
Пример:

```
int m;
cin>>m;
int *mas2= new int[m];
```

После работы с динамическим массивом необходимо очистить память. Синтаксис:

```
delete []имя;
```

Пример:

```
delete []mas2;
```

Динамические массивы не инициализируются при создании!
Пример: см. задачу 2 https://replit.com/@IekatierinaLat1/Arrays#main.cpp

Алгоритмы работы с массивами

Классификация:

1. Алгоритмы простого обхода:
    1. заполнение, ввод, вывод;
    2. подсчет (сумма, произведение, количество и т.д.);

2. Алгоритмы поиска:
    1. линейный (последовательный) поиск образца, минимального, максимального, и т.д.;
    2. бинарный поиск;
3. Алгоритмы сортировки:
    1. устойчивая сортировка, сохраняющая предыдущий порядок элементов;
    2. неустойчивая сортировка, не гарантирующая сохранение предыдущего порядка элементов;
4. Прочие.

Пример: алгоритм подсчета суммы. См. задачу 3 https://replit.com/@IekatierinaLat1/Arrays#main.cpp

Алгоритмы поиска

Задача поиска элемента в массиве при заданном условии имеет интуитивно-понятный алгоритм решения, заключающийся в последовательном просмотре всех элементов массива с одновременной проверкой условия поиска. В зависимости от условия поиска алгоритмы можно условно разделить на две категории — когда искомый элемент гарантированно найдется в массиве, и когда его может в массиве не оказаться.

Алгоритм поиска (максимального) минимального элемента

1. Максимальный или минимальный элементы в массиве есть всегда. Даже когда все элементы равны между собой, минимальным или максимальным можно считать любой из них.
2. Алгоритм. Сначала считаем минимальным самый первый (элемент с номером 0), потом последовательно просматриваем все остальные элементы. Если рассматриваемый элемент меньше минимального, то будем считать минимальным его.
3. Для этих алгоритмов часто требуется знать не только значение максимального или минимального элемента, но и его номер.

Пример: см. задачу 4 https://replit.com/@IekatierinaLat1/Arrays#main.cpp

Поиск образца

Поиск образца — алгоритм поиска некоторого элемента (введенного с клавиатуры или полученного каким-то другим способом) в массиве. Часто называют линейным (последовательным) поиском. Если такой элемент есть, часто нужно вывести не только подтверждение этого факта, но и его номер. Если же элемент отсутствует, необходимо об этом сообщить.
Для удобства перед началом обхода массива переменной, в которой должен быть сохранен номер искомого элемента, часто присваивают значение, которого заведомо не может быть среди номеров элементов. Таким является, например, любое отрицательное число. Чаще всего используют -1. Далее последовательно просматривается весь массив и, если элемент будет найден, значение номера искомого элемента меняется на текущее. Дальше необходимо досрочно завершить цикл. Если этого не сделать и в массиве будет несколько одинаковых элементов, то будет найден последний из них, причем неэффективным способом.
При поиске элемента в вещественном массиве нужно не забывать, что вещественные числа нельзя сравнивать напрямую.

Пример: см. задачу 5 https://replit.com/@IekatierinaLat1/Arrays#main.cpp

Бинарный поиск

При использовании линейного алгоритма поиска каждый раз приходится полностью заново просматривать весь массив. Есть ли какой-либо способ выполнять эту задачу быстрее? Ответ на этот вопрос будет положительным, но только при одном условии — если поиск будет осуществляться в

упорядоченном массиве. Для простоты дальше будем считать, что массив **упорядочен по возрастанию**. Для упорядоченных по убыванию массивов решение будет аналогичным.

1. Находим середину массива и сравниваем элемент, стоящий посередине, с искомым. Если они равны, элемент найден.
2. Если искомый элемент меньше стоящего в середине, то меняем правую границу массива на номер стоящего в середине элемента.
3. Если искомый элемент больше стоящего в середине, то меняем левую границу массива на номер стоящего в середине элемента.
4. Повторяем шаги 1-3, пока не найдем элемент или пока не убедимся, что его нет.

Пример: см. задачу 7 https://replit.com/@IekatierinaLat1/Arrays#main.cpp (можно рассмотреть после примера 6, т.к. массив должен быть предварительно упорядочен)

Примечание: условие предварительной сортировки массива может показаться странным с практической точки зрения. Действительно, данные редко хранят в отсортированном виде. Однако в базах данных очень часто используют индексацию — специальный инструмент, который позволяет считать таблицу как бы упорядоченной по соответствующему столбцу. В одной и той же таблице может быть несколько индексов по разным полям. Поиск с использованием индексов по смыслу будет аналогичен бинарному поиску и будет работать существенно быстрее последовательного поиска.

Алгоритмы сортировки

Сортировка предназначена для упорядочивания массивов по какому-либо признаку или совокупности признаков. Чаще всего сортировка выполняется по возрастанию или убыванию. В некоторых случаях используется более сложное условие сортировки, например сортировка по возрастанию модулей элементов.

Устойчивость сортировки

Устойчивой называется сортировка, сохраняющая предыдущий порядок элементов. Если говорить об одномерном массиве, то на практике разницы между устойчивым и неустойчивым алгоритмами не будет. Но при сортировке таблиц с данными устойчивость часто очень важна.

Допустим, есть некоторый список абитуриентов, участвующих в конкурсе. Всем удобно, когда список отсортирован в алфавитном порядке. Однако после того, как абитуриенты сдадут экзамены, возникает необходимость в ранжированном списке — их нужно отсортировать в порядке убывания суммарного балла. Возникает вопрос, что будет с теми абитуриентами, у которых суммарный балл одинаков? При использовании устойчивого алгоритма сортировки алфавитный порядок для людей с равными баллами сохранится. Если же применить неустойчивый алгоритм, то сохранение этого порядка не гарантируется. Поэтому устойчивость считается очень важной характеристикой алгоритмов сортировки.

В настоящее время существует множество различных алгоритмов сортировки. Рассмотрим некоторые из них.

Сортировка выбором

Сложность алгоритма $O(n^2)$. Подробнее про сложность см. далее. На практике немного быстрее сортировки пузырьком за счет меньшего количества обменов. Суть алгоритма: на каждом шаге выбираем минимальный элемент в массиве и меняем его с текущим. На следующем шаге рассматриваем только элементы, которые стоят после текущего.

```
3 -4 1 0 7 -2 6
-4 3 1 0 7 -2 6
-4 -2 1 0 7 3 6
-4 -2 0 1 7 3 6
-4 -2 0 1 7 3 6
-4 -2 0 1 3 7 6
-4 -2 0 1 3 6 7
```

Зеленым выделена уже отсортированная часть массива.
Черным — текущий элемент.
Красным — минимальный элемент в остатке массива (в 4 строчке совпадает с текущим).
Синим — неотсортированный остаток массива, где производится поиск минимального элемента.
Пример: см. задачу 6 https://replit.com/@IekatierinaLat1/Arrays#main.cpp

Быстрая сортировка

Один из самых быстрых алгоритмов сортировки массивов, разработана Т. Хоаром и иногда называется сортировкой Хоара.. Неустойчива (не сохраняет предыдущий порядок). Сложность O(n*log(n)). Суть алгоритма:

1. Выбрать из массива опорный элемент. Например, для случайного массива можно выбирать элемент, стоящий посередине.
2. Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на две части. В одной части должны находиться элементы, меньшие опорного, в другой большие или равные ему.
3. Для обеих частей массива выполнять рекурсивно те же действия, пока размер каждой из частей больше единицы.

Графическая иллюстрация алгоритма (источник)

Пример: см. задачу 8 https://replit.com/@IekatierinaLat1/Arrays#main.cpp

Сортировка слиянием* (материал повышенной сложности)

Сортировка слиянием (merge sort) — алгоритм сортировки, использующий принцип разделяй и властвуй. Массив разбивается на более мелкие части, для которых применяется тот же алгоритм, пока части не станут достаточно маленькими и отсортированными. Далее происходит слияние отсортированных частей.

Алгоритм является устойчивым, хорошо распараллеливается и кешируется. Алгоритм достаточно эффективно работает на структурах последовательного доступа (связные списки, файлы и т.д.). Сложность алгоритма O(n*log(n)).

К недостаткам можно отнести нечувствительность к почти отсортированным массивам, а также необходимость дополнительного использования памяти, по объему равной размеру массива.

Пример: https://replit.com/@IekatierinaLat1/Recursion