Поразрядные операции и их применение в алгоритмах

Маской (maska) называем переменную (или значение), которая содержит двоичный код и применяется для изменения определенных битов в 1 или 0.

x<<n	Сдвиг влево двоичного кода (умножение на 2^n)	unsigned int x=7; x=x<<2; результат 0x0000001C
x>>n	Сдвиг вправо двоичного кода (деление на 2^n)	unsigned int x=28; x=x>>2; результат =0x00000007
x & maska	Поразрядное И (применяется для записи в указанный разряд значения 0)	Правило выполнения операции 111 & 100 = 100 Установить в двоичном коде переменной ***x*** только 9-ый справа бит в 0 unsigned short int x=0xAEFF; unsigned short int maska=0xFDFF; x=x & maska результат 0xACFF
X | maska	Поразрядное ИЛИ (применяется для записи в указанный разряд 1)	Правило выполнения операции 111 | 100 111 Установить в двоичном коде переменной x 9-ый справа бит в 1 unsigned short int x=0xACFF; unsigned short int maska=0x0200; x=x | maska результат 0xAEFF;

X ^ maska	Исключающее ИЛИ для поразрядных операций Используется для проверки соответствующих битов двух переменных, если они имеют разные значения, то результат 1, а если равны, то 0.	Правило выполнения операции
		1111
		^ 0001
		= 1110
		unsigned int x=0xF, a=1;
		a=x^a;
		Результат: в переменной ***a*** значение 0x0000000E

~	Инверсия (0 заменяет на 1, а 1 на 0)	x=0x0F; ~x; результат 0xF0

1. Выполнить упражнения

Записать выражение, которое решает задачу:

1) установить 7-й бит переменной x в 1;
2) установить 5-й и 3-й биты в значения переменной x в 0;
3) инвертировать 5-й бит переменной x.
4) Даны два целых положительных числа X и Y одной разрядности. Написать выражение, которое формирует новое значение X, где n битов, начиная с позиции p, установлены так, как n самых правых битов в Y, а остальные биты не изменились.

Модель значения X: xxx….xnnnnx….xxx

Модель значения Y: yyy……ynnnny

2. Какую задачу реализует код функция?

```
void f(unsigned int x)
{
        int n=sizeof(int)*8;
        unsigned  maska=(1<<(n-1));
        for(int i=1; i<=n;i++)
        {
                s=s+(x&maska)>>(n-i));
                maska=maska>>1;
        }
}
```

3. Определите задачу, которую решает функция set. Приведите подробное описание реализации решения. Приведите тест для каждой функции.

```
#define BITSPERWORD 32
#define SHIFT 5
#define maska 0x1F
#define N 10000000
int a[1+N/ BITSPERWORD];
void set(int i){ a[i>> SHIFT] |=(1<<(i & maska ));}
```

Пример процесса разработки программы с выбором структуры представления данных для оптимизации сортировки

Рассмотрим пример, демонстрирующий выбор структуры данных, для обеспечения всех требований предъявляемых к разрабатываемой программе.[1].

1.1. Понимание предложенной задачи

Программист (назовем его Первым программистом) должен был разработать программу сортировки большого объема данных, хранящихся в файле. Программа нужна через сутки. При выборе алгоритма сортировки он решил посоветоваться с более опытным программистом (назовем его Вторым программистом). И у них состоялся диалог, в котором Второй программист уточнял особенности решения задачи.

Первое предложение Второго программиста – это использовать алгоритм сортировки слиянием, который применяется для сортировки данных в файлах.

Но Первому программисту не хотелось углубляться в теорию алгоритмов.

Тогда Второй программист предложил использовать готовую программу, опубликованную в литературе по программированию, эта программа имеет 200 строк кода, разбитого на 12 функций. На написание и отладку такой программы Первый программист потратил бы не больше недели. Но первый программист сомневался, что проблема будет решена.

Тогда Второй программист понял, что существуют какие-то требования к программе и он стал расспрашивать второго программиста о его задаче.

В результате Второй программист выяснил следующее:

- Разрабатываемая программа будет встроена в большую программу, поэтому она не может использовать системные средства.
- Файл содержит 10^7 записей. Записи – это целые положительные семизначные числа. Узнав размер файла, Второй программист сразу предложил отсортировать записи в оперативной памяти, так как их не так много. Но, Первый программист ответил, что хотя на компьютер оснащен большим объемом памяти, программа сортировки является лишь малой частью большой программы. К моменту ее выполнения будет свободным лишь 1Мб памяти.
- Программа должна сортировать по возрастанию записи файла приблизительно раз в час, т.е. время сортировки должно быть минимальным и составлять секунд десять.

В результате обсуждения программисты сформулировали постановку задачи.

После этого была сформулирована точная постановка задачи.

[1] Жемчужина программирования

1.2. Точная постановка задачи

Дано. Файл, содержащий не более n ($n=10^7$) целых положительных чисел, каждое из которых семизначное число, т.е. принадлежит диапазону [1000000..9999999] и среди них нет повторяющихся.

Результат. Упорядоченный по возрастанию список чисел, сохраненных в файле.

Ограничения:

- Доступной оперативной памяти 1МБ.
- Дисковая память неограниченна.
- Время выполнения программы 10 секунд.

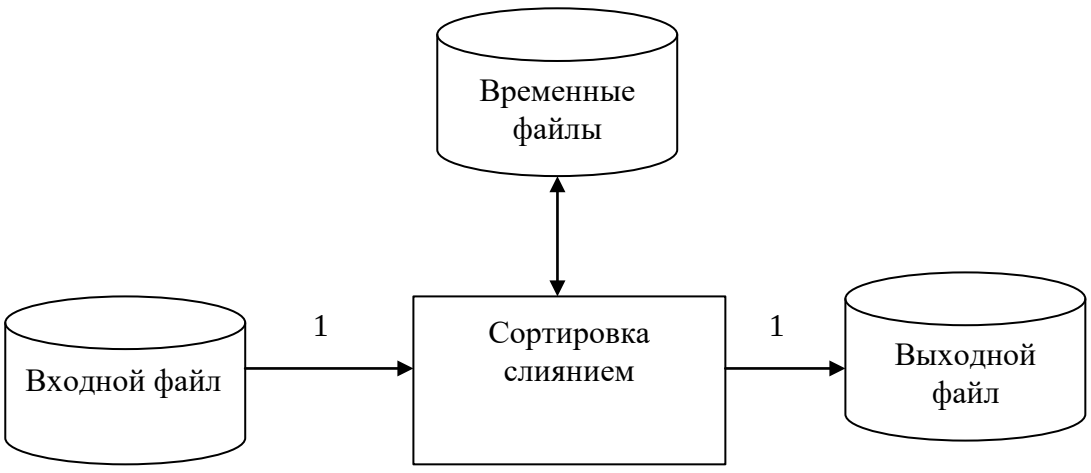
1.3. Исследование разработки

Рассмотрение возможных вариантов решения

Первый вариант – Однопроходный алгоритм.

Использовать программу сортировки слиянием из 200 строк кода, оптимизировав ее с учетом того, что сортируются семизначные числа. Это сократит размер кода до нескольких десятков строк. Но на кодирование и отладку потребуется несколько дней. Сортировка слиянием для файлов требует использование двух временных файлов. Сортировка слиянием считывает входной файл один раз и разливает его по нескольким временным файлам (два файла), сортирует его с использованием временных файлов, которые считываются и записываются много раз. Выполнение операций чтения и записи во временные файлы осуществляется за длительное время, так как внешняя память имеет низкое быстродействие выполнения операций.

Схема однопроходного алгоритма



Второй вариант – Сорока проходной алгоритм.

Данный подход опирается на использование деталей конкретной задачи.

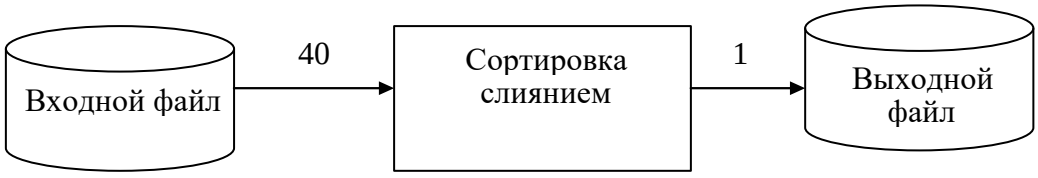
а) Если представить числа как последовательности символов, то на каждое число

потребуется семь байт. Тогда в доступном 1Мб памяти поместится всего 143 000 чисел.

б) Если для представления чисел использовать ячейки из 32 бит, то в 1Мб поместится уже 250 000 номеров. Тогда программа может сортировать записи файла в оперативной памяти, считывая его за 40 раз (40*250 000=107): т.е. за первое считывание считываются записи с номерами с 0 по 249 999, которые затем сортируются и записываются в выходной файл. За второй проход сортируются записи с номерами 250 000 по 499 999 и т.д. Для сортировки записей можно использовать алгоритм быстрой сортировки (алгоритм Хоара), который занимает всего 20 строк кода. Вся программа тогда будет содержать около 50 строк кода.

Схема Сорока проходного алгоритма

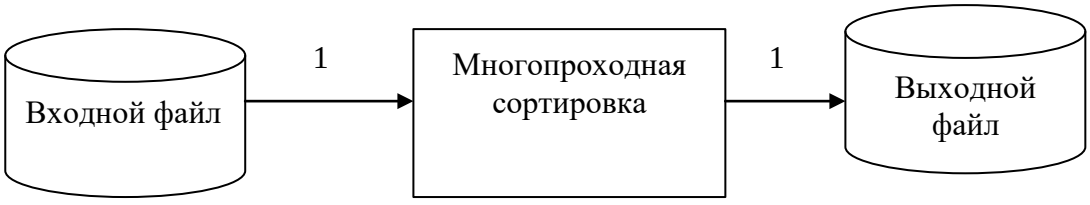
Считывание входного файла порциями по 250 000 чисел осуществляется 40 раз, результат записывается в выходной файл за один раз, не используя временных файлов, поэтому время сортировки будет меньше, чем в первом варианте.



Третий вариант - битовая реализация

Оптимальной была бы программа, работающая по схеме, соединяющей преимущества предыдущих: входной файл считывается один раз и временные файлы не используются.

Схема оптимального алгоритма.



Такую схему можно реализовать, когда все данные при сортировке будут размещены в 1Мб оперативной памяти. Т.о. задача сводится к отображению 10^7 целых семизначных чисел приблизительно в восемь миллионов доступных бит (1Мб=1024*1024*8 бит=8 388 608 бит а надо 10 000 000 битов, т.е. недостает еще около 2 000 000 битов).

1.4. Неформальный алгоритм

Использование последовательности бит для представления чисел напрашивается само собой. Суть такого представления в следующем. Пусть имеется 20 целых чисел и их можно значения в диапазоне от 1 до 20 и их надо представить 20 битами. Последовательность чисел, вставляемая в битовую последовательность из 20 бит будет такой {1, 2, 5, 9, 12, 8}. Сначала вся последовательность бит заполнена нулями. Биты в последовательности нумеруются с нуля.

Теперь создадим отображение, которое вставляет 1 в бит с номером, равным значению вставляемого числа, , тогда получим следующую битовую последовательность: 01100100110010000000.

В решаемой задаче файл представим как строку из десяти миллионов битов (программист нашел недостающие 2 000 000 битов), в которой i-ый бит равен 1, если в файле присутствует число со значением i.

Такой подход возможен в задаче именно с такими исходными данными: числа целые и нет повторяющихся, и их сравнительно не много, чтобы все представить битами в памяти.

При использовании битового массива для представления сортируемых чисел, программу можно представить как последовательность из трех задач:

1. Инициализация массива нулевыми значениями.
2. Считывание целых чисел из файла и установка в 1 соответствующих бит.
3. Формирование упорядоченного выходного файла путем последовательной проверки бит массива и вывод в файл номеров тех бит, которые установлены в 1.

1.5. Определение алгоритмов выбранного решения

Имена объектов алгоритма:

n – количество битов в массиве. bit[n] – битовый массив

Абстрактное описание алгоритма

Алгоритм на псевдокоде

```
//Часть 1: инициализация массива битов нулями for i←0 t n
    bit[i]:=0;

//Часть 2: Заполнение битового массива значениями Чтение числа из файла в переменную i bit[i]:=1
//Часть 3: Формирование упорядоченного выходного файла for i←0 t n
    if bit[i]=1 then записать i в файл;
```

Этого наброска программы оказалось достаточно, чтобы программист решил задачу. Как видно из данного материала, что представление данных в виде последовательности бит позволило выполнить все ограничения поставленной задачи.

Выводы

Внимательное изучение задачи позволило программистам найти правильный подход к решению.

Битовые структуры позволили экономно использовать память. Битовые структуры используются для представления плотного набора элементов конечного множества, в который элемент может входить не более одного раза, и, где с элементами не связано более ни каких дополнительных данных.

Использование битового массива привело к сокращению времени выполнения программы. Это произошло за счет того меньшее количество данных требует меньшего времени для обработки, а во-вторых, хранение данных в оперативной памяти, а не на диске сокращает затраты на обращение к ним.

Программа получилась достаточно простой. Простота влечет надежность и эффективность программы, их проще создавать и сопровождать.

При разработке алгоритма использовался подход, основанный на следующих принципах:

1. Понимание предложенной задачи
2. Постановка абстрактной задачи
3. Неформальное описание программы

«Некоторые программисты размышляют несколько минут, а потом программируют целый день вместо того, чтобы подумать час, а потом час программировать»[2]

Неформальное описание программы – это исследование пространства разработки (самой задачи). Неформальные языки высокого уровня помогают описывать проекты программ: псевдокод определяет последовательность выполнения программы, а абстрактные типы данных представляют основные структуры. Решение одной задачи может иметь несколько вариантов, которые неформально описаны.

Неформальное описание программы может показать, что одно из решений гораздо предпочтительнее прочих. Реализовать код надо по возможности просто и прямолинейно, используя самые мощные и доступные средства.

«Задача инженерии программного обеспечения в том, чтобы регулировать сложность, а не увеличивать ее»[3].

[2] Жемчужина программирования

[3] Памелу Зейв