[1]Катков В.Л., [2]Новоселова А.Н.

[1]Объединенный институт проблем информатики НАН Беларуси, Минск, katkov@newman.bas-net.by, [2]Белорусский государственный университет, Минск, novoselova@bsu.by

КАЧЕСТВО КОМПЬЮТЕРНЫХ ПРОГРАММ НАПРЯМУЮ ЗАВИСИТ ОТ КАЧЕСТВА ОБУЧЕНИЯ

Аннотация. В жизненном цикле разработки программного обеспечения ЭВМ ключевым моментом является постановка задачи, определяющая ее будущие основные характеристики, такие как время счета, расход памяти, надежность программы и т.д. Это требует от разработчика программы провести тщательный предварительный анализ постановки задачи. В статье на простых примерах показано, что вдумчивый подход к постановке задачи и ее реализации в виде компьютерной программы позволяют значительно сокращать время счета и зачастую приводят к значительно более эффективному и надежному программному коду. Отмечается, что, к сожалению, этому главному вопросу жизненного цикла разработки программ уделяется недостаточное внимания в учебных программах высших учебных заведений, в которых львиная доля времени отводится на типовые схемы программирования и почти ничего не рассказывается о возможных альтернативах программы.

Abstract. In the life cycle of the software development the key moment is a problem definition from which follows its future basic characteristics, such as run time, an economical expenditure of resources, first of all - memories, program reliability, and so on. All this demands from the software developer the preliminary analysis of the statement problem definition. In article on simple examples it is shown, that the careful analysis allows to reduce considerably run time and frequently leads to much more effective and reliable program code. It is marked, that unfortunately, to this main question of life cycle program development it is given insufficient attention in curriculums of higher educational institutions in which the significant part of time is allocated on typical schemes of programming and almost nothing is spoken about possible alternatives of the created program.

Введение. Ввиду чрезвычайно широкого толкования термина "программирование" под "программистом" мы будем подразумевать специалиста, разрабатывающего программное обеспечение ЭВМ, имеющего подготовку в объеме вузовской специальности "прикладная математика" или близкой к ней, и опыт работы по специальности не менее двух лет. В последние годы в связи с массовым распространением программирования в качестве инструмента "ширпотреба" наблюдается тревожная тенденция падения качества программ, вследствие чего значительно растут затраты на эксплуатацию вновь "испеченного" программного продукта, прежде всего - времени счета программы.

Как известно, жизненный цикл программного обеспечения охватывает значительный период времени, начиная с этапа формулировки требований и кончая ее опытной эксплуатацией и сопровождением [1 - 3]. Одним из центральных моментов, обеспечивающих высокое качество программы, является этап постановки задачи, когда будущая программа обретает "скелет" и закладываются ее основные характеристики. Неслучайно известный специалист по информатике Д. Кнут назвал свою серию книг, посвященных разработке программ, "Искусством программирования" [4].

Не имея возможности в рамках короткой статьи охватить все этапы жизненного цикла программы, остановимся на одном из них - этапе постановки задачи, от которого в решающей степени зависит качество создаваемой программы.

Пример 1. Сосчитать значения интеграла $J_n = \int_0^1 x^n e^{x-1} dx$ для $n =$ 0, 1, ..., 20. Начинающий программист сразу же бросается писать про-

грамму, используя первый попавшийся под руку метод, например, метод трапеций. Получившийся текст будет иметь примерно такой вид:

```
double h=0.01; // Шаг сетки узлов
double s, x, f0, f1; int n;
void main(void)
{ for (n=0; n<=20; n++)
  { s=0; x=0; f0=1-exp(-1);
    while (x<=1)
    { x+=h; f1=pow(x,n)*exp(x-1); s+=0.5*h*(f0+f1); f0=f1; };
    // Печать результата: n и Jn
  }
}
```

Программа занимает 10 строк. Кроме сомнительного способа вычисления x^n с помощью библиотечной функции pow(x,n), программа может считаться приемлемой (с некоторыми оговорками) и требует лишь подбора шага интегрирования h. Эксперименты показывают, что шаг h = 0.01 обеспечивает точность 2 – 3 верные цифры после запятой.

Однако существует более эффективный способ решения задачи. Опытный программист, изучая постановку задачи, стремится получить дополнительную информацию, полезную для ее решения и разработки машинного варианта алгоритма. Легко сообразить, что результат счета должен быть положительным, т.к. интеграл берется от положительной функции. Кроме того, $J_{n+1} < J_n$, поскольку $x^{n+1}<x^n$ на [0, 1], т.е. значения интеграла должны монотонно убывать с ростом n.

Интегрированием по частям легко получить рекуррентную формулу: $J_n = \left[x^n e^{x-1}\right]_0^1 - n\int_0^1 x^{n-1}e^{x-1}dx = 1 - nJ_{n-1}$. Начальное значение последо-

вательности $\{J_n\}$ при $n = 0$ легко вычисляется аналитически: $J_0 = 1 - 1/e$.

Вся программа теперь умещается в одну строку:

```
J[0]=1-exp(-1); for (n=1; n<=20; n++) J[n]=1-n*J[n-1];
```

Однако результаты счета оказываются обескураживающими. Вначале значения интеграла уменьшаются с ростом n, как и должно быть, а затем начинают стремительно расти, поочередно меняя знак (таблица 1).

Таблица 1 – Вычисление J_n по рекуррентной формуле $J_n = 1 - nJ_{n-1}$

n	0	2	4	6	8	10	12	14	15	16
J_n	0.632	0.264	0.171	0.127	0.101	0.084	0.072	0.121	-0.818	14.1

В чем причина такого странного поведения решения? Причина - в вычислительной неустойчивости алгоритма! В рекуррентной формуле J_n=1-nJ_{n-1} большое значение n умножается на маленькое значение J_{n-1}, что приводит к факториальному росту погрешности. Можно ли преобразовать формулу так, чтобы с ростом n погрешность не увеличивалась? Оказывается, можно! Для этого следует обратить счет: начинать с большого значения N, полагая $B_N = 0$, и находить B_{n-1} по формуле $B_{n-1} = (1-B_n)/n$. Параметр N подбирался экспериментально; оказалось, что результаты счета при $N = 40$ и $N = 80$ совпадают с восемью знаками после запятой.

Таблица 2 – Вычисление B_n по рекуррентной формуле $B_{n-1} = (1 - B_n)/n$

n	0	2	4	6	8	10	12	14	15	16
J_n	0.632	0.264	0.171	0.127	0.101	0.084	0.072	0.121	-0.818	14.1
B_n	…	…	…	…	…	…	0.072	0.063	0.059	0.056

Мы обозначили через B_n результаты, получающиеся по "обращенной" формуле. В таблице 2 многоточием отмечены значения B_n, совпадающие с соответствующими значениями J_n.

Итак, на этапе постановки задачи удалось значительно упростить алгоритм и сократить длину программы.

***Пример* 2.** Сосчитать число "счастливых" билетов, пронумерованных натуральными числами. Билет считается счастливым, если сумма n левых цифр (d_1+…+d_n) совпадает с суммой n правых цифр (d_{n+1}+…+d_{2n}). Используются десятичные цифры 0, 1, …, 9, n – параметр задачи (рисунок 1).

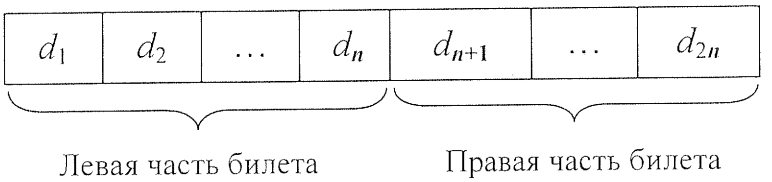

Рисунок 1 – Схематичный вид номера билета

Очевидный способ решения задачи состоит в прямом переборе всех билетов (их будет 10^{2n}) и получении для каждого из них суммы n левых и n правых цифр, после чего счетчик числа счастливых билетов увеличивается на 1, если "левая" и "правая" суммы совпали. Небольшие размышления о способах перебора билетов и их структуре подсказывают, что время решения задачи можно значительно сократить, если использовать половинный перебор. Пусть N_k - число "полубилетов" с суммой цифр, равной k. Например, шестиразрядных полубилетов (n = 3) с суммой цифр, равной 2, будет 6 штук, т.е. N_2 = 6. (Все такие полубилеты перечислены на рисунке 2). Полных же билетов с суммой цифр, равной k, будет N_k^2, а общее число счастливых билетов составит $A=\sum_{k=0}^{9n} N_k^2$. Вычисление величины A тре-

$n=3$	$k=2$
002	002
…	…
011	011
…	…
020	020
…	…
101	101
…	…
110	110
…	…
200	200

Рисунок 2 – Подсчет числа шестиразрядных счастливых билетов

бует всего 10^n операций.

Итак, предварительный анализ алгоритма решения задачи дает ускорение счета на несколько порядков.

***Пример* 3.** Найти два максимальных элемента в массиве $\{A_1, A_2, \ldots, A_N\}$. Тривиальный вариант алгоритма – двукратный просмотр массива – требует $2N$ операций сравнения.

Можно предложить более эффективный способ, требующий $\frac{3}{2}N$ операций. Для этого массив делится на две (необязательно равные) части, и в каждой из половин отыскивается максимальное число. Пусть это будут M_1 для первой половины и M_2 – для второй. Далее поступаем следующим образом:

Если ($M_1 > M_2$) // Первый локальный максимум больше второго, то

{ Положить $Max_1 = M_1$, а M_2 заслать в 1-ю половину A на место M_1;

Найти максимум в 1-й половине и выдать его в качестве Max_2;

Завершить работу;

};

иначе // Второй локальный максимум больше первого. Необходимо

{ Положить $Max_1 = M_2$, а M_1 заслать во 2-ю половину A на место M_2;

Найти максимум во 2-й половине и выдать его в качестве Max_2;

Завершить работу;

};

Итак, за $\frac{1}{2}N + \frac{1}{2}N$ операций сравнения находятся два максимума в двух половинах массива A и еще за $\frac{1}{2}N$ операций отыскивается максимум в первой либо во второй половине массива A, т.е. решение задачи требует $\frac{3}{2}N$ операций сравнения вместо $2N$. Экономия времени в 25 % может составить заметную абсолютную величину, если этот фрагмент

программы стоит внутри цикла.

Заключение. Показано, что предварительный анализ постановки задачи может существенно повысить качество программы, прежде всего – уменьшить время ее исполнения. Для повышения качества программирования необходимо в учебных планах вузов существенно усилить разделы, связанные с технологией программирования, с акцентом на автоматизированной поддержке всех этапов жизненного цикла программы и привлечения новых инструментов, повышающих надежность и эффективность программ. Например, это могут быть препроцессоры, проверяющие соответствие физических размерностей величин [5].

Список использованной литературы

1. Ван Тассел Д. Стиль, разработка, эффективность, отладка и испытания программ: Пер. с англ. – М.: Мир, 1981. – 320 с.
2. Майерс Г. Надежность программного обеспечения: Пер. с англ. – М.: Мир, 1980. – 360 с.
3. Katkov V.L., Novosselova A.N. Intelligent Tutoring in Numerical Methods. - Proc. of AI-ED World Conf. on Artificial Intelligence in Education. Kobe, Japan, 1997, p. 607-608.
4. Кнут Д. Искусство программирования, том 1. Основные алгоритмы = The Art of Computer Programming, vol. 1. Fundamental Algorithms. – 3-е изд. М.: "Вильямс", 2006. – С. 720.
5. Катков В.Л., Новоселова А.Н. Программная инженерия и аналитические выкладки. Информационные системы и технологии (IST'2009) = International systems and (IST'2009) technologies: материалы V междунар. конф.-форума (Минск, 16-17 нояб. 2009 г.). В 2 ч. Ч. 2. /редкол.: Н.И. Листопад [и др.]. – Минск: А.Н. Вараксин, 2009. – С. 74 - 77.