О ГРАМОТНОЙ АЛГОРИТМИЗАЦИИ

Ермаков Илья Евгеньевич
ООО «Метасистемы», ТИ ОГТУ, Орёл, ermakov@metasystems.ru
Рюмшин Борис Валерьевич
ООО «Метасистемы», ОГУ, Орёл, rbv@phys-math.ru

Доклад посвящен методам систематического построения алгоритмов и преподаванию этих методов.

The paper contains some summary about systematic design of algorithms and teaching this methods in education.

Базовые понятия. На наш взгляд, очень удачное введение в понятие алгоритма содержится в пособии Дейкстры «Краткое введение в искусство программирования», гл. 1 [1]. С самого начала должна быть показана связь алгоритма (модели поведения системы) и процесса (конкретной реализации поведения системы); иерархическая организация алгоритмов и процессов (декомпозиция и абстракция). Такое понимание близко методическим курсам в традициях А.П. Ершова, основанным на учебных исполнителях.

Первый этап — учебный исполнитель. Особая задача — построение первой части учебного курса. Учащимся предстоит освоение множества новых понятий. Желательно как только возможно разделить эти понятия друг от друга, сделать их доступными для изучения по отдельности и постепенно складывать из них единую технику построения программ (заметим, что именно так ведётся постановка базовой техники в спорте или музыке).

На начальном этапе необходимо избежать работы с данными вообще (т.е. исключить понятия ввода-вывода, вычисления, переменных...), а найти способ изучить и закрепить в чистом виде понятия, связанные с планированием действий — т.е. алгоритмическим управлением. Для этого А.П. Ершовым и его последователями была предложена в 70-х гг. концепция учебных исполнителей. Учащиеся строят алгоритмы, управляющие

исполнителем в некой обстановке. Эта идея получила развитие в школьном учебнике коллектива Мехмата МГУ: А.И. Кушниренко и др. [2], который издавался в 90-х гг., но был недооценён в массе педагогов. Мы в своих курсах во многом опираемся на идеи названного учебника, а так же на университетский учебник тех же авторов [3].

Нами разработан алгоритмический тренажёр, который позволяет писать на Компонентном Паскале в среде BlackBox алгоритмы, управляющие учебным исполнителем Робот, полностью аналогичным учебной системе КуМир. Основные методические задачи этой части курса: проектирование алгоритмических процессов «сверху вниз» - разбиение на вспомогательные процедуры и активное их использование; алгоритмы с «обратной связью» - планирование действий в зависимости от обстановки, в которой находится исполнитель (операторы WHILE и IF). В целом же не будет преувеличением сказать, что первейшая задача — научить пользоваться и «чувствовать» цикл WHILE (и тем самым предотвратить «скатывание» к общепринятому безграмотному построению циклов на основе FOR-BREAK).

Алгоритмы с переменными. Для студентов программистских и математических специальностей имеет смысл вводить понятие переменной и присваивания «чистым образом», без связи с понятием ячеек памяти, чтения-записи и т. п. (однако такой подход, по нашему опыту, может не сработать на инженерных специальностях).

«Приучение» к переменным выполняется постепенно. Сначала — переменные и однократные присваивания, с пониманием переменной как обозначения для результата некоторого шага вычислений. Затем демонстрируется цикл, переопределяющий значение переменной многократно — расчёт числовых последовательностей. Таким образом, показывается циклический процесс, порождающий последовательность значений (в общем случае — векторов). Вектор переменных цикла проходит эту последовательность от начала до конца.

Наконец, демонстрируются присваивания, содержащие по обе стороны

одну переменную (очевидно, что они имеют смысл только внутри циклов) — рассматриваются циклы, порождающие рекуррентные последовательности (начиная от сумм, степеней, факториалов — и кончая сколь угодно сложными циклами, варьирующими многими величинами). **Формируется понимание цикла как итерационного процесса, при котором вектор переменных сходится от начальных значений к результату (который является решением уравнения в предикатах — постусловия цикла).**

Базовые схемы циклов. Здесь мы опираемся на формулировки координатора Информатики-21 Ф.В. Ткачёва, из его спецкурса на физическом факультете МГУ [4].

Схема «полный проход».

Имеем некую последовательность - элементов, шагов, ситуаций… Длина последовательности не известна, но известно условие проверки её окончания. Тогда может быть организован цикл «с обратной связью» (после совершения шага проверяем, не достигнута ли ещё цель).

```
взять_первую_ситуацию;
WHILE ~конец_ситуаций (* т.е. "удалось взять очередную ситуацию" *) DO
  (* Предусловие: имеем текущую ситуацию, подлежащую обработке *)
  полезное_действие_в_текущей_ситуации;
  переход_к_следующей_ситуации
END
```

Если в задаче наблюдается семантика полного прохода, целесообразно строго следовать приведённой схеме — навык видеть и автоматически безошибочно её применять должен быть сформирован упражнениями.

Пример с потоковым чтением. Правильно организованный интерфейс потокового чтения предоставляет функцию «считать очередной элемент» и признак, который показывает, было ли считывание успешным.

```
reader.Read(elem);
WHILE reader.Done() DO
  ...работаем с elem...
  reader.Read(elem)
END
```

Очевидно, что попыток чтения производится на одну больше, чем оборотов цикла с полезным действием. И «дублирование» возникает именно по этой причине; любые попытки его устранить не приведут к прояснению ситуации, а только к запутыванию и потере прозрачности базовой схемы.

Тем более, что в других случаях начальное действие и переход могут быть разными:

```
it := items;
WHILE it # NIL DO
  ...работаем с it...
  it := it.next
END
```

Действия «взять_начальную», «перейти_к_следующей», не говоря про само полезное действие, могут быть сколь угодно необычными, самое главное — узнать саму семантику полного прохода, и без фантазий следовать шаблону.

Схема «линейный поиск»

Как и в предыдущей схеме, имеем последовательность «чего угодно». Задача — найти такую ситуацию, которая удовлетворяет условию поиска, или показать, что её нет (достигнут конец последовательности).

```
взять_первую_ситуацию;
WHILE ~конец_ситуаций & ~( ... условие поиска ..) DO
    ... возможно, действие над текущей ситуацией ...
  взять_следующую_ситуацию
END;
IF ~конец_ситуаций THEN
  ... нашли, делаем что надо, с той ситуацией, на которой остановился цикл...
ELSE
  ... не нашли, делаем что-то, если нужно
END
```

Для такой выразительной записи линейного поиска важно, чтобы & вычислялся сокращённо, т.к. условие поиска может не иметь смысл в случае конец_ситуаций.

Пример с «узнаванием» линейного поиска и полного прохода

(из студенческой программы)

Имеется граф потока вычислений - т.е. ориентированный граф без циклов, вершина — это операция, по дугам поступают аргументы. Его можно расчитывать в режиме потока данных (распространяя «волну» от аргументов к результатам), а можно «лениво» (от интересующего нас конечного узла запрашивая рекурсивно аргументы). Если считаем в первом режиме, то возникает «фронт волны» (или ещё говорят - «рабочая стопка», workpile). На каждом витке workpile-алгоритма мы ищем в стопке те узлы графа, для

которых уже готовы все аргументы.

Нужно написать для узла графа функцию ReadyForCalc(node): BOOLEAN. Понимающий студент сразу видит, что это <=> «для любого непосредственного предка *node* выполнено *defined*». Т.е. **A anc : anc IN Ancestors(node) : anc.defined**, где A — квантор всеобщности, A. Точно так же он знает, что это эквивалентно **~ (E anc : anc IN Ancestors(node) : ~anc.defined**, т.е. «среди предков не существует такого, для которого ~*defined*». Т.е. чтобы доказать, что «для любого…» нужно построить «опровергающий» линейный поиск обратного условия и вернуть отрицание результата этого поиска. Это может отображаться в следующий исходный текст (граф хранится на квадратной матрице, так, как его вводят в редакторе — для простоты, чтоб не ставить студенту задачу авторазмещения графа на листе):

```
PROCEDURE ReadyForCalc (g: Flow.Grid; ref: Flow.Ref): BOOLEAN;
  VAR j: INTEGER; cell: Flow.Cell;
BEGIN
  (* A cell : cell IN Anc(g[ref]) : cell.defined *)
  (* ~ ( E cell : cell IN Anc(g[ref]) : ~cell.defined *)
  (* т.е. линейный поиск ячейки с ~cell.defined. RETURN поиск_не_удался *)
  ASSERT (g[ref.x, ref.y].argCount > 0, 20);
  cell := g[ref.x, ref.y];
  j:= 0;
  WHILE (j < cell.argCount) & g[cell.args[j].x, cell.args[j].y].defined DO
    INC(j);
  END;
  RETURN j = cell.argCount
END ReadyForCalc;
```

Задача проверки выполнения какого-то условия для всех ситуаций встречается достаточно часто, её надо узнавать и по такой схеме трансформировать в «опровергающий» линейный поиск.

Наконец, и сам цикл продвижения волны по графу тоже сводится к шаблонному полному проходу, надо только понять, какая имеется последовательность ситуаций. За ситуацию примем набор необсчитанных ещё вершин, хранящихся в рабочей стопке (фронте). Т.е. инвариантом цикла будет *(* INV: в front находятся ещё не обсчитанные вершины, ~defined *)*.

При этом очевидно, что целесообразно хранить в стопке

непосредственно только те вершины, которые имеют шанс стать обсчитанными на очередном витке.

Определим две операции: *Calc* ищет в стопке те узлы, которые *ReadyForCalc* и выполняет расчёт их значений. *Step* ищет в стопке обсчитанные вершины (они там оказываются в середине витка цикла, когда инвариант временно нарушен после *Calc*), выбрасывает их из стопки, а взамен помещает их непосредственных последователей (тех, которые ещё туда не попали). Тогда имеем полный проход:

```
поместить_в_стопку_начальные_аргументы;
Step;
WHILE количество_в_стопке > 0 DO
    Calc;
    Step
END
```

Как обычно, с помощью первой части тела цикла мы продвинулись вперёд, нарушив соотношение-инвариант, с помощью второй — восстановили инвариант, после чего идёт проверка «ещё не достигли цели?».

Большинство встречающихся на практике циклов имеют, в общем, семантику полного прохода или линейного поиска. В случае более специфической семантики (цикл варьирует вектор величин в соответствии с некоторым сложным соотношением) необходимо применение рассуждений, основанных на формулировке инварианта цикла. Основой грамотной алгоритмизации является работа Э. Дейкстры «Дисциплина программирования» [5] (которую можно назвать «дифференциальным исчислением» для программирования), однако для большинства случаев достаточно её частной проекции - показанного выше стиля рассуждений.

Для наработки и закрепления навыков грамотного построения алгоритмов можно рекомендовать задачник Касьянова и Сабельфельда [6].

«Профессиональное» программирование. В сфере ИТ сложилась парадоксальная ситуация, когда подавляющее большинство «профессионалов» (в основном самоучек, хлынувших в «лёгкую» отрасль) не

владеют методами систематического проектирования программ. (Приведём ссылку на показательное обсуждение на форумах OberonCore [7]). Умение проектировать циклы — только вершина айсберга, но оно отлично может служить индикатором культуры мышления программиста. Достаточно посмотреть на общераспространённый линейный поиск через FOR-BREAK; программистов же, владеющих методами инварианта и многоветочного цикла Дейкстры, можно «заносить в красную книгу». Чтобы понять серьёзность проблемы, представим, что большинство инженеров-строителей не имело бы ни малейшего понятия о сопромате. Невладение методологией обычно сопровождается оправданиями, которые в основном сводятся к следующим: «математические доказательства слишком громоздки и неприменимы к большим программам» или «эти теории далеки от практики; какая разница, как строить алгоритм, если всё равно основные проблемы на уровне проектирования архитектуры...».

Первое возражение только лишний раз демонстрирует незнание вопроса, т. к. в «анализе по Дейкстре» предполагается не доказательство уже написанных алгоритмов, а изначальное обоснованное их проектирование, с опорой на ясные рассуждения в духе показанных выше. Степень формальности рассуждений должна выбираться исходя из необходимости (по принципа - «доказать — это значит представить соображения в таком виде, чтобы их правильность была очевидна всем, владеющим соотв. аппаратом»); основная идея здесь — не формальные выкладки, а строгие рассуждения, позволяющие уверенно вывести алгоритм из общих соображений. При таком подходе время и усилия на программирование не увеличиваются, а уменьшаются; качество и понятность результата не сопоставимы с хаотичным стилем программирования.

На второе возражение следует сказать, что, во-первых, качество всей «постройки» неизбежно складывается из качества каждого отдельного «кирпича», а во-вторых, грамотная алгоритмизация — только вершина айсберга, показатель владения систематическим мышлением (в противовес

«народно-ополченческому»). Сложность разработки ПО можно уменьшить в первую очередь выносом основных проектных решений на уровень выше, чем программирование — на уровень системного анализа и математизированных моделей. В этом случае программа строится просто как проекция соображений такого высокого уровня в программные конструкции.

Отдельно стоит отметить миф о том, что математически обоснованным является исключительно функциональное программирование. Что это не так, демонстрирует опыт советской (Новосибирск) и европейской (Цюрих) школ программирования - опыт систематического императивного программирования. Донести некоторые его срезы до читателя — одна из целей этой статьи. Если нам удалось показать, что проектирование императивных алгоритмов не имеет никакого отношения к «перекладке значений из одних ячеек памяти в другие», то эту цель можно считать достигнутой.

Ссылки *(чтобы получить названные книги в электронном виде, можно обратиться к авторам статьи).*

1. Э. Дейкстра. Краткое введение в искусство программирования. EWD316, 1971. Пер. А.С. Деревянко - http://khpi-iip.mipk.kharkiv.edu/ library/ extent/ dijkstra/ ewd316/ index316.html

2. А.Г. Кушниренко, Г.В. Лебедев, Р.А. Сворень. Основы информатики и вычислительной техники. М.: 1991.

3. А.Г. Кушниренко, Г.В. Лебедев. Программирование для математиков. М.: 1988.

4. Конспект одной из лекций Ф.В. Ткачёва - http://www.inr.ac.ru/~info21/08.pdf

5. Э. Дейкстра. Дисциплина программирования. М.: 1978.

6. В.Н. Касьянов, В.К. Сабельфельд. Сборник заданий по практикуму на ЭВМ. М.: 1986.

7. Обсуждение на форуме - http://forum.oberoncore.ru/viewtopic.php?p=28238