Российская академия наук
Сибирское отделение
Институт систем информатики
имени А. П. Ершова

Е. В. Бодин, Л. В. Городняя, Н. В. Шилов

ПО КАКОМУ ПРЕДМЕТУ ОЛИМПИАДА?

Препринт
126

Новосибирск 2005

Работа посвящена обсуждению ситуации с олимпиадами по информатике и программированию. Основная проблема — это многообразие по сравнению с другими предметными олимпиадами и, в то же время, некоторая путаница в определении предмета и формы проведения таких олимпиад. Основные определения предмета «информатика» и сложность его преподавания в школе показаны на примере серии решений довольно известной задачи, нередко в разных вариантах включаемой в комплекты олимпиадных задач. Алгоритмы решения этой задачи, включая эвристический алгоритм Э.Дейкстры, иллюстрируют неоднозначность критериев оценки решений и разнообразие построений, дающих формально правильный результат.

Работа поддержана грантом РФФИ № 05-07-90162.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

Е. В. Бодин, Л. В. Городняя, Н. В. Шилов

WHAT IS THE SUBJECT OF THE COMPUTER SCIENCE CONTEST?

Preprint
126

Novosibirsk 2005

The paper is devoted to the current situation with Informatics and Programming contests. The major problem consists in the relative multitude of formats of programming contests as compared to contests in other fields; moreover, there is a certain fuzziness in the definition of the subject and organization structure. The basic definitions of the subject of Informatics and the complexity of its teaching at schools are demonstrated in a series of solutions of a rather well-known problem, which is often included, with slight variations, into such contests. The solution algorithms for this problem, including a heuristic algorithm of E.Dijkstra, illustrate the ambiguity of problem evaluation criteria and the variety of constructions yielding a formally correct solution.

ВВЕДЕНИЕ

Работа посвящена обсуждению ситуации с олимпиадами по информатике и программированию. Основная проблема — это многообразие по сравнению с другими предметными олимпиадами и, в то же время, некоторая путаница в определении предмета и формы таких олимпиад.

Основные определения предмета «информатика» и сложность его преподавания в школе показаны на примере серии решений довольно известной задачи, нередко в разных вариантах включаемой в комплекты олимпиадных задач. Речь идет о задаче, известной как проблема начальника порта. Алгоритмы решения этой задачи, включая эвристический алгоритм Э.Дейкстры, иллюстрируют неоднозначность практичных критериев оценки решений и разнообразие построений, дающих формально правильный результат.

Многообразие олимпиад по информатике и программированию не случайно. Отчасти они выполняют функцию дополнительной системы образования, реагирующей на отставание основной системы от потребностей сегодняшнего дня. Олимпиадное программирование побуждает всех его участников — и школьников, и учителей, и организаторов конкурсов — к углубленному пониманию предмета, к активному овладению методами программирования, к творческому созданию обстановки, способствующей проявлению и развитию талантов.

Приведём только краткий обзор олимпиад школьного уровня по Новосибирской области, о которых нам известно из личного опыта.

1. ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ

Дело в том, что противопоставление информатики и программирования — пример противопоставления части и целого в том случае, когда эта часть имеет особое значение по сравнению с другими частями. Близкий по смыслу пример — химия и органическая химия. Конечно же, органическая химия – часть химии вообще, но как по количеству органических соединений, так по их сложности и роли в жизни человека, это, безусловно, очень «привилегированная» часть химии. Примерно так же обстоит дело с информатикой и программированием.

Информатика — это наука и отрасль производства, которая занимается проблемами построения «информационных» моделей мира и производства,

методами хранения и обработки[1] новой информации из этих моделей. Программирование — это часть информатики, которая занимается реализацией на компьютерах алгоритмов обработки информации. Таким образом, за рамками программирования остаются вопросы построения информационных моделей и неалгоритмические аспекты хранения и обработки информации.

Для старшеклассников существует Всероссийская (личная) олимпиада, которая проводится уже более 15-ти лет. Она организована как последовательность этапов отбора участников для включения в сборную России, делегируемую на международные олимпиады. После школьной идут районные, городские, областные, окружные или зональные и российский туры, задача которых — поиск школьников, способных показать высокие результаты при решении и программировании алгоритмических задач, предлагаемых на международных чемпионатах. Эта работа по отбору сопровождается летними и зимними сборами для особо перспективных школьников.

Лет пять назад появилась новая форма школьных олимпиад для старшеклассников — Всероссийская командная олимпиада, основная задача которой программирование алгоритмических задач. Кроме выявления и развития индивидуальных способностей, эта олимпиада ставит цель выявить и развить навыки работы в команде, когда над одной задачей работает сразу несколько человек, разбив её на подзадачи, или разбив решение задачи на независимые этапы. Есть еще школьная секция Всесибирской командной олимпиады, задуманной как альтернатива схеме Всероссийской командной олимпиаде, включающая не только алгоритмизируемые задачи. Всесибирская олимпиада поощряет творческую изобретательность в сочетании с практичностью решения сложных задач.

Для среднего и старшего школьных звеньев создаются олимпиады по информационным технологиям (с использованием офисных, графических и/или математических пакетов). Они становятся все более традиционными, обретают официальный статус и по классу задач постепенно сближаются с главной Всероссийской олимпиадой. Так с 2004 г. Microsoft Research объявила новую конкурсную линию (алгоритмика, графика, мультимедиа и проекты), которую в Новосибирске начинают осваивать.

Для этой же возрастной группы с начала 90-х годов XX века в Новосибирске функционирует система разноплановых номинаций открытого конкурса «Молодые информатики Сибири», учрежденная Высшим Колледжем

[1] Мы не будем здесь обсуждать, что такое информация, компьютер и алгоритм, а оставим эти вопросы философам.

Информатики Новосибирского государственного университета. Среди номинаций есть как бескомпьютерные (нацеленные на разработку и анализ моделей и алгоритмов), так и программистские (направленные на программную реализацию алгоритмов).

Для школьников младшего и среднего звеньев проводятся олимпиады по алгоритмике с использованием так называемых «исполнителей» и языка Logo. Начинались они как всероссийские олимпиады по схеме «Кенгуру», но постепенно стали местными олимпиадами для Новосибирской области.

Исходя из нашего опыта, можно утверждать, что, начиная с середины 90-х гг., в олимпиадах по информатике на школьном и вузовском уровнях в нашей стране утвердился массовый переход на регламент международных командных студенческих олимпиад по программированию International Collegiate programming Contest (ICPC[2]), которые проводятся под эгидой Association for Computing Machinery (ACM[3]). Этот регламент сочетает автоматическую (без вмешательства судейской коллегии) проверку готовых программ на системе тестов (заранее подготовленной судейской коллегией) и автоматический хронометраж времени, понадобившегося на каждую решённую задачу (т.е. вплоть до выхода готовой программы). При таком подходе задача считается решённой, если программа правильно справилась со всеми тестами. При подведении итогов участники олимпиады сначала ранжируются по числу решённых задач, а потом — по суммарному времени, понадобившемуся на все решённые задачи[4].

По нашему мнению, есть несколько причин массового перехода на регламент ACM ICPC, среди которых есть и объективные, и субъективные. Некоторые из них мы обсудим ниже.

Можно сказать, что главный субъективный фактор — это высокий общественный авторитет ACM ICPC (а формат ICPC — это автоматическая проверка готовых программ на системе тестов). Подтверждением авторитета ACM ICPC в России стала встреча 28 мая 2004 г. в Кремле Президента России В.В. Путина с российскими победителями и призерами финала ICPC 2004 г. А 26 января 2005 г. В.В. Путин подписал указ «О присуждении премий Президента Российской Федерации в области образования за 2003 год». Среди 15 работ, отмеченных премиями, есть коллективный труд «Разработка концепции и создание организационной структуры, учебно-методического и программного обеспечения инновационной системы под-

[2] См. сайт олимпиады ICPC на http://icpc.baylor.edu/icpc/.

[3] См. сайт организации ACM на http://www.acm.org/.

[4] Ещё учитывается число попыток сдать каждую из решённых задач.

готовки высококвалифицированных кадров в области информационных технологий» [5]. По сути дела, названная концепция есть результат огромной работы по организации и проведению в нашей стране соревнований ACM ICPC, которую ее создатели по собственной инициативе и практически на общественных началах проделали в период 1994–2004 гг.

Рис. 1. Фото пресс-службы Президента России (http://www.kremlin.ru): Москва, Кремль. Встреча с победителями и призерами студенческого чемпионата мира 2004 г. по программированию

По-видимому, есть два главных объективных фактора перехода на автоматическое тестирование готовых программ:

[5] Авторский коллектив: д.т.н. профессор, , профессор, декан факультета информационных технологий и программирования СПбГУ ИТМО В.Г. Парфенов, д.т.н., профессор СПбГУ ИТМО В.Н. Васильев, ассистенты кафедры компьютерных технологий СПбГУ ИТМО Р.А. Елизаров и А.С. Станкевич, к.ф.-м.н., декан математико-механического факультета Уральского госуниверситета им. А.М. Горького М.О. Асанов, старший преподаватель кафедры системного программирования СПбГУ Н.Н. Вояковская, д.ф.-м.н., профессор, ректор Алтайского государственного технического университета им. И.И. Ползунова В.В. Евстигнеев, к.т.н., доцент МИФИ В.М. Кирюхин, д.ф.-м.н. профессор, проректор МГУ им. М.В. Ломоносова А.В. Михалев, к.ф.-м.н., доцент кафедры математической кибернетики и компьютерных наук Саратовского госуниверситета им. Н.Г. Чернышевского А.Г. Федорова.

– программа является аккумулированным итогом всего цикла решения задачи, включая построение моделей, разработку алгоритма и его реализацию;
– машинное тестирование позволяет без вмешательства человека отсеять программы, которые не соответствуют условиям задачи.

На наш взгляд, кроме субъективных и объективных факторов были ещё и причины социального характера, которые в 90-х годах привели к массовому переходу на автоматическое тестирование готовых программ на олимпиадах по информатике в нашей стране. Действительно, известные экономические трудности переходного периода в России в этот период не позволяли полномасштабно финансировать многие виды образовательной деятельности вообще и олимпиады по информатики в частности. Многое в этой области деятельности держалось просто на энтузиазме одиночек и одобрении руководства народным образованием в стране, регионах и конкретных учебных заведениях, которым по крохам всё-таки удавалось выкраивать средства на проведение олимпиад. Поэтому в то время достаточно единодушно (хотя и без восторга) была воспринята методика автоматического тестирования готовых программ на олимпиадах по информатике, которая минимизирует финансовые и трудовые затраты по сравнению с проверкой вручную моделей, алгоритмов и стиля программирования в задачах, конечная цель которых — работающая программа.

Однако только автоматическое тестирование готовых программ и автоматический хронометраж времени, потребовавшегося на подготовку правильных программ, может оказаться недостаточным для корректной оценки решений даже программистских задач на олимпиадах по информатике. Для того чтобы аргументировать этот тезис, мы предлагаем пример олимпиадной программисткой задачи, которую будем называть «проблемой начальника порта»[6].

2. ПРОБЛЕМА НАЧАЛЬНИКА ПОРТА

Эта проблема формулируется следующим образом.

> На рейде порта с N причалами находится ровно N кораблей. Начальник порта получает штормовое предупреждение и должен назначить каждому из кораблей его индивидуальный при-

[6] Задача использовалась на очном туре Открытого конкурса «Молодые информатики Сибири» в 1996 г. в программисткой номинации.

чал. Положения всех кораблей в момент получения штормового предупреждения известны. Положения причалов фиксированы и известны. Непосредственно в этот момент никакое препятствие (особенности береговой линии, расположение других причалов или положение других кораблей на рейде) не мешает любому из кораблей двигаться прямо к любому из причалов. Но во избежание столкновений, маршруты кораблей не должны пересекаться. Помогите начальнику порта распределить корабли по причалам.

Теперь давайте по порядку разберемся с геометрической и информационной моделями проблемы начальника порта.

Геометрическая модель проблемы — это N чёрных и N белых точек на плоскости, соответствующих положениям кораблей на рейде и расположению причалов, а возможные (гипотетические) маршруты — отрезки прямых между белыми и чёрными точками. Заметим кстати, что никакие три из этих точек не являются коллинеарными (т.е. не лежат на одной прямой). Нам необходимо построить (если это вообще возможно) N попарно непересекающихся отрезков, у каждого из которых один конец — это некоторая белая точка, а другой — это некоторая чёрная точка.

Простейшая информационная модель содержит всю возможную геометрическую информацию о положении N чёрных и N белых точек в виде массивов их координат. Для размещения в памяти компьютера этой информации достаточно 4N ячеек памяти (по одной ячейке на каждую из двух координат каждой из 2N точек). В этой модели по доступной информации можно проверить пересекаются ли гипотетические маршруты, расстояния между кораблями на рейде, расстояния между причалами, расстояния между причалами и кораблями и т.д. Но нам требуется выбрать (если это вообще возможно) множество из N пар вида (белая точка, чёрная точка), которым соответствует N попарно непересекающихся отрезков, концы которых выбранные пары точек.

Целевой «объект» геометрической модели проблемы начальника порта — множество из N отрезков с концами (разного цвета) в заданных N чёрных и N белых точках, причём, не содержащее пересекающихся отрезков. Поэтому имеет смысл ввести специальный тип данных СОЕДинение, значения которого — всевозможные множества из N отрезков с концами (разного цвета) в заданных N чёрных и N белых точках. Среди всех значений типа СОЕД нас интересуют такие множества отрезков, которые не содержат пересечений. Поэтому кажется вполне оправданным ввести

булевскую функцию ХОР(X: СОЕД): BOOL на значениях этого типа, которая возвращает значение TRUE тогда и только тогда, когда множество отрезков X не содержит пересекающихся отрезков. Заметим, что тип СОЕД состоит из N! различных значений: первая белая точка может быть соединена с любой из N чёрных точек, вторая белая — с любой из оставшихся (N-1) чёрных точек, и т.д. Поэтому можно считать, что перед нами перечислимый тип, среди значений которого есть первый элемент, доступный благодаря константе ПЕРВ: СОЕД, и есть функция следования СЛЕД(X: СОЕД): СОЕД, которая по каждому элементу (кроме последнего), переданному в качестве значения фактического параметра, возвращает следующий элемент.

В терминах типа данных СОЕД геометрическая модель проблемы начальника порта интерпретируется следующим простым способом: среди всех возможных значений типа СОЕД найти (если есть) такой, который удовлетворяет ХОР. Эта переформулировка позволяет спроектировать чисто переборный алгоритм решения проблемы начальника порта[7]:

```
VAR X: СОЕД;
VAR C, N: INT;

      X:= ПЕРВ ;
      C:=0 ;

  WHILE   C<N! & ~ХОР(X)
      DO
         X:= СЛЕД(X);
         C:=C+1
      OD ;

  IF   C=N!
         THEN OUTPUT (‘SORRY!’)
          ELSE OUTPUT (X)
```

Роль счётчика C в этом алгоритме очевидна: он подсчитывает, сколько соединений из N! возможных уже проверено, и позволяет завершить цикл как только все они проверены и ни одного «хорошего» не найдено.

[7] Здесь и далее «!» обозначает факториал, а «~» — булевское отрицание.

Достоинство такого алгоритма — простота описания, а главный недостаток — высокая оценка сложности (так как происходит перебор всех возможных значений типа СОЕД). Отметим, однако, что во-первых, данный алгоритм никак не использует факт, что никакие три точки не являются коллинеарными, и во-вторых, данный алгоритм найдёт хорошее соединение тогда и только тогда, когда оно существует[8].

С точки зрения реализации этот алгоритм хорошо поддерживается информационной моделью. Действительно, значения типа СОЕД можно хранить в виде пары массивов координат ЧЕР и БЕЛ длины N, подразумевая, что первый отрезок соединяет точку ЧЕР[1] с точкой БЕЛ[1], второй отрезок — точку ЧЕР[2] с точкой БЕЛ[2] и т.д.; функция ПЕРВ соответствует вводу черных и белых точек, функция СЛЕД — перестановке элементов массива БЕЛ с помощью (например) алгоритма Э. Дейкстры генерации следующей перестановки, а функция ХОР — проверке на отсутствие пересечения для всех пар отрезков [ЧЕР[I], БЕЛ[I]] , [ЧЕР[J], БЕЛ[J]] при $1 <= I < J <= N$.

Отнюдь не оптимистические результаты анализа эффективности алгоритмов перебора (пропорциональной N!) наталкивают на мысль попробовать применить какую-нибудь эвристику. В данном случае для повышения эффективности речь идёт о локальном устранении пересечений в множестве из N отрезков с разными концами в чёрных и белых точках, в надежде, что это позволит в конце концов устранить все пересечения.

Очевидно, имеет смысл использовать тот же специальный тип данных СОЕД, что и в алгоритме полного перебора, с той же булевской функцией ХОР(X: СОЕД): BOOL, среди значений которого есть первый элемент, доступный благодаря константе ПЕРВ: СОЕД. Но теперь вместо функции следования СЛЕД(X: СОЕД): СОЕД, которая позволяет осуществить перебор всех значений этого типа, будет использоваться функция ЩЁЛК(X: СОЕД): СОЕД. Эта функция реализует желаемую эвристику, локально устраняя пересечения отрезков, т.е. выбирает некоторую пару (если такая существует) пересекающихся отрезков $[B',W'] \cap [B'',W''] \neq \varnothing$ из множества отрезков X и «переЩЁЛКивает» их, заменяя на пару отрезков [B',W''] и [B'',W'], т.е. возвращает множество отрезков $(X \setminus \{[B',W'] , [B'',W'']\}) \cup \{[B',W''], [B'',W']\}$, как это показано на рис. 2.

[8] Следовательно заранее не известно, остановится ли алгоритм с ответом в виде конкретного множества из N отрезков между чёрными и белыми точками без пересечений, или остановится с неутешительным ответом ‘SORRY!’.

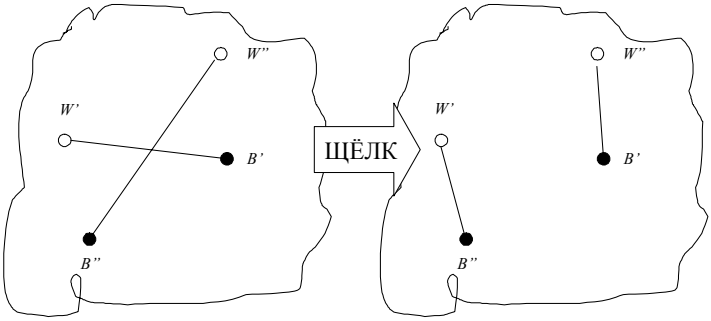

Рис. 2. Операция ЩЁЛК на типе СОЕД

Теперь мы можем описать алгоритм эвристического перебора для проблемы начальника порта:

```
VAR X: СОЕД;
VAR C, N: INT;

   X:= ПЕРВ;
   C:=0;

WHILE   C<N! & ~ХОР(X)
     DO
        X:= ЩЁЛК(X);
        C:=C+1
     OD;
IF  C=N!

   THEN OUTPUT (‘SORRY!’)
   ELSE OUTPUT (X)
```

Фактически этот алгоритм эвристического перебора повторяет алгоритм полного перебора, но использует вместо функции следования, перебирающей все возможные соединения, функцию локального устранения конфликта, которая перебирает только некоторые соединения, получающиеся в результате перещёлкивания отрезков. Значения счётчика С в этом алгоритме уже не позволяют отслеживать, все ли соединения проверены, но позволяют избежать зацикливания (так как всех соединений ровно N!, и следовательно, период функции ЩЁЛК не более N!).

Достоинства эвристического алгоритма — простота описания и явно использованная правдоподобная эвристика. Главные недостатки этого алгоритма — это, с одной стороны, высокая оценка сложности (так как не исключён перебор всех возможных значений типа СОЕД), а, с другой стороны, возможная «неполнота», т.е. алгоритм может не найти хорошее соединение даже тогда, когда оно существует (так как не исключено, что функция ЩЁЛК не перебирает все значения типа СОЕД, а «циклит» по значениям, которые содержат пересекающиеся отрезки). Отметим также в очередной раз, что данный алгоритм пока никак не использует факт, что никакие три точки не являются коллинеарными.

С точки зрения реализации эвристический алгоритм так же хорошо поддерживается информационной моделью, как и алгоритм полного перебора. Мы уже обсуждали выше, как можно реализовывать значения типа СОЕД и функцию ХОР на основе массивов координат черных и белых точек ЧЕР[1..N] и БЕЛ[1..N]. Теперь добавим, что функция ЩЁЛК может использовать ту же пару пересекающихся отрезков [ЧЕР[I], БЕЛ[I]] , [ЧЕР[J], БЕЛ[J]] с 1 <= I < J <= N, которую «обнаружила» реализация функции ХОР, и возвращать соединение, в котором белые точки БЕЛ[I] и БЕЛ[J] переставлены местами. Таким образом, с точки зрения человека, которому надо программировать, эвристический алгоритм проще, чем алгоритм полного перебора, так как перемена местами двух элементов массива — это всего три строчки кода, а генерация следующей перестановки — это уже целая подпрограмма.

Возможная неполнота эвристического алгоритма наводит на мысль исследовать, когда всё-таки функция ЩЁЛК не зацикливается на значениях типа СОЕД, содержащих пересекающиеся отрезки. Другими словами, когда следующий вариант эвристического алгоритма без использования счётчика всё-таки решает нашу проблему:

```
VAR X: СОЕД;
      X:= ПЕРВ;

WHILE  ~ХОР(X)
     DO
       X:= ЩЁЛК(X)
     OD;

OUTPUT (X)
```

Этот вариант эвристического алгоритма решения проблемы начальника порта предложил в 1994 г. в публичной лекции классик науки программирования Э. Дейкстра[9]. Отметим очевидный факт, что если этот алгоритм завершает работу, то он выдаёт решение проблемы начальника порта. Удивительным является то, что этот алгоритм гарантированно останавливается и, следовательно, является полным алгоритмом решения проблемы!

Для доказательства завершаемости (т.е. полноты) своего эвристического алгоритма Э. Дейкстра применил так называемый метод потенциалов, разработанный другим классиком науки программирования Р. Флойдом. В простейшей форме метод потенциалов для доказательства завершаемости алгоритма может быть сформулирован следующим образом:

- выбирается конечное множество числовых значений, которые называются потенциалами;
- каждому набору значений используемых переменных сопоставляется некоторый потенциал.

Тогда, если каждое исполнение тела любого цикла в алгоритме уменьшает значение потенциала, то алгоритм завершает работу[10].

Применительно к своему эвристическому алгоритму Э. Дейкстра рассуждал следующим образом. Для любого значения X типа СОЕД примем в качестве его потенциала P(X) сумму длин отрезков, которые входят в X: $P(X)= \Sigma_{[B,W] \in X} |B,W|$. Так как множество значений типа СОЕД конечно, то и

[9] К сожалению, эта лекция не опубликована и не доступна в электронном архиве Э. Дейкстры на `http://www.cs.utexas.edu/users/EWD/`.

[10] Корректность метода легко установить от противного. Действительно, если алгоритм всё-таки зацикливается, то тело некоторого из его циклов исполняется бесконечно много раз подряд. Но каждое исполнение тела каждого цикла приводит к уменьшению значения потенциала. Следовательно, если алгоритм зацикливается, то существует бесконечная убывающая последовательность значений потенциалов. Это противоречит конечности множества потенциалов.

соответствующее множество потенциалов тоже конечно. Далее, тело единственного цикла эвристического алгоритма Э. Дейкстры состоит из единственного оператора присваивания X:= ЩЁЛК(X), а исполнение этого тела уменьшает значения потенциала. Это следует из рис. 3, так как в силу неравенства треугольника[11], а также теоремы, гласящей, что сумма диагоналей прямоугольника больше суммы двух его сторон:

$$|B'W''| + |B'',W'| < |B'W'| + |B',W'|$$

и, следовательно,

$$P(X) = (\Sigma_{[B,W] \in X} |B,W|) = (|B',W'| + |B'',W''|) + (\Sigma_{[B,W] \in X\setminus\{[B',W'],[B'',W'']\}} |B,W|) >$$
$$> (|B',W''| + |B'',W'|) + (\Sigma_{[B,W] \in X\setminus\{[B',W'],[B'',W'']\}} |B,W|) =$$
$$= (\Sigma_{[B,W] \in ЩЁЛК(X)} |B,W|) = P(ЩЁЛК(X)).$$

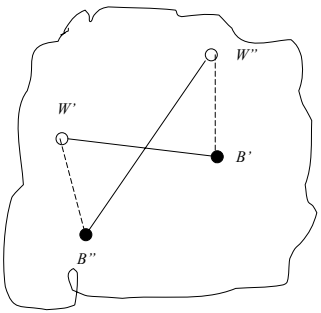

Рис. 3. Причины изменения потенциала при операции ЩЁЛК

Поэтому, в соответствии с методом Р. Флойда, эвристический алгоритм Э. Дейкстры всегда останавливается и является полным алгоритмом решения проблемы начальника порта.

Достоинства эвристического алгоритма Э. Дейкстры — изящество представления и полнота решения исходной проблемы, а также простота реализации. Главным недостатком этого алгоритма остаётся высокая оценка сложности, так как по-прежнему не исключён перебор всех возможных значений типа СОЕД. Отметим также, что при обосновании полноты этого

[11] Неравенство треугольника применимо, ибо в соответствии с геометрической моделью проблемы начальника порта ни какие три точки не лежат на одной прямой.

алгоритма явно используется факт, что никакие три точки не являются коллинеарными.

3. ЭФФЕКТИВНЫЕ АЛГОРИТМЫ

Переборный характер интуитивно ясных алгоритмов (полного перебора и эвристического перебора) подвигают на поиск эффективного в терминах геометрической модели алгоритма решения проблемы начальника порта.

Естественно, может возникнуть идея попробовать, не приведёт ли к успеху стратегия «разделяй и властвуй» проектирования эффективных алгоритмов. «Разделять» в нашем случае надо множество из N чёрных и N белых точек на плоскости прямой линией так, что бы в одной полуплоскости осталось [N/2] чёрных и [N/2] белых точек, и в другой полуплоскости осталось (N – [N/2]) чёрных и (N – [N/2]) белых точек; такое разделение прямой сводит задачу для N чёрных и N белых точек к двум задачам о (примерно) (N/2) чёрных и (N/2) белых точках.

К сожалению, нам не известно, как построить такую прямую. Однако, если никакие три точки не лежат на одной прямой, то возможно построить прямую, которая проходит через одну из чёрных и одну из белых точек и разбивает плоскость так, что в левой полуплоскости будут содержаться некоторое число $K \in [0..(N-1)]$ чёрных и такое же число белых точек, а в правой полуплоскости будут содержаться все остальные (N – K – 1) чёрных и столько же белых точек[12]. Такое разделение прямой сводит задачу для N чёрных и N белых точек к двум задачам: одна — о K чёрных и K белых точках, вторая — о (N – K – 1) чёрных и (N – K – 1) белых точках.

Для того чтобы описать геометрический (вернее, топологический) алгоритм построения методом разбиения «хорошего» соединения для заданных N чёрных и N белых точек на плоскости (среди которых нет коллинеарных троек), нам пригодятся следующие типы данных: тип для точек ТЧК и отрезков ОТР, тип для множеств точек МН_ТЧК и для множеств отрезков МН_ОТР. На значениях типов МН_ТЧК и МН_ОТР определены обычные теоретико-множественные константа для пустого множества (∅), операции объединения (∪) и новая операция мощности множества МОЩ(X: МН_ОТР∪МН_ТЧК): INT (мы будем использовать это обычное математическое обозначение |X| вместо МОЩ(X)). На значе-

[12] Доказательство этого геометрического факта не входит в наши цели и мы оставляем его читателю.

ниях типов ОТР и ТЧК определена функция МНЖ(X: ОТР∪ТЧК): МН_ОТР∪МН_ТЧК, которая возвращает одноэлементное множество {X} (мы будем использовать это обычное математическое обозначение {X} вместо МНЖ(X)), а на значениях типа ТЧК определена функция ОТР(X: ТЧК, Y: ТЧК): ОТР, которая возвращает отрезок [X,Y] (мы будем использовать это обычное математическое обозначение [X,Y] вместо ОТР(X,H)). Кроме того, на значениях типа ТЧК и МН_ТЧК определены функции ЛЕВ(X: ТЧК, Y: ТЧК, S: МН_ТЧК): МН_ТЧК и ПРВ(X: ТЧК, Y: ТЧК, S: МН_ТЧК): МН_ТЧК, которые возвращают подмножество всех точек из S, находящееся левее прямой (X, Y) и, соответственно, правее этой прямой.

В этих терминах геометрический алгоритм можно представить следующим образом.

```
VAR B, W: МН_ТЧК;

FUNCTION  CP(B, W: МН_ТЧК) : МН_ОТР;
     VAR X, Y: ТЧК;
  BEGIN
    IF B=W=∅
        THEN CP:= ∅
        ELSE
    FOR X∈B, Y∈W
      DO
        IF |ЛЕВ(X,Y,B)|=|ЛЕВ(X,Y,W)| & |ПРВ(X,Y,B)|=|ПРВ(X,Y,W)|
            THEN BEGIN CP:= {[X,Y]} ∪
                                    CP(ЛЕВ(X,Y,B),ЛЕВ(X,Y,W)) ∪
                                    CP(ПРВ(X,Y,B),ПРВ(X,Y,W)) ;
                          BREAK
                 END
      OD;
   END;

BEGIN
  OUTPUT('введите множество из N чёрных точек') ;
     INPUT (B) ;
  OUTPUT('введите множество из N белых точек') ;
    INPUT (W) ;
  OUTPUT(CP(B,W))
END.
```

Даже грубая оценка[13] позволяет ограничить сложность этого алгоритма кубической функцией от N. Корректность и полнота описанного геометрического алгоритма следуют из факта о существовании «разбивающей» прямой, который был сформулирован перед алгоритмом (доказательство оставлено читателям). Следовательно, геометрический алгоритм является эффективным и полным алгоритмом решения проблемы начальника порта.

С точки зрения реализации этот алгоритм так же хорошо поддерживается информационной моделью. Так, например, любое значение типа МН_ОТР можно хранить в виде пары массивов ЧЕР и БЕЛ координат точек (длины N) и счётчика K ∈ [0..N] числа элементов в этом множестве отрезков, понимая, что эта тройка представляет множество из K отрезков [ЧЕР[1], БЕЛ[1]], … [ЧЕР[K], БЕЛ[K]]. Аналогично, значения типа МН_ТЧК можно хранить в виде одного массива координат точек (длины N) и счётчика числа точек K ∈ [0..N]. Достаточно трудоёмкой представляется реализация операций ЛЕВ и ПРВ. Для точек P и Q и множества точек S надо составить уравнение прямой, проходящей через точки P и Q в виде a × x + b × y = 0 с a > 0, а затем перебирать точки R из S, подставлять их координаты в (a × x + b × y) и вычислять значение этого выражения: если это значение меньше 0, то точка R попадает в ЛЕВ, если больше 0, то R попадает в ПРВ.

Пожалуй, главный недостаток геометрического алгоритма состоит в том, что он не переносится непосредственно с евклидовой плоскости на многомерное евклидово пространство, т.е. не пригоден для решения следующей геометрической задачи:

> В D-мерном (D>1) Евклидовом пространстве заданы N черных и N белых точек, среди которых никакие три не лежат на одной прямой. Требуется построить (если это вообще возможно) N попарно непересекающихся отрезков, у каждого из которых один конец — это некоторая белая точка, а другой — это некоторая чёрная точка.

---

[13] Для любых значений B и W типа МН_ТЧК, и любых значений X и Y типа ТЧК, если X ∈ B и Y ∈ W, то |ЛЕВ(X, Y, B)| < |B|, |ЛЕВ(X, Y, W)| < |W|, |ПРВ(X, Y, B)| < |B| и |ПРВ(X, Y, W)| < |W|. Поэтому глубина рекурсии в описанном алгоритме не более N. На каждом этаже этой рекурсии в цикле перебирается не более N^2 пар точек X и Y, и для первой пары из них, такой что |ЛЕВ(X, Y, B)| = |ЛЕВ(X, Y, W)| & |ПРВ(X, Y, B)| = |ПРВ(X, Y, W)|, происходит спуск на следующий этаж рекурсии.

Напротив, оба рассмотренных переборных алгоритма пригодны для решения этой геометрической задачи, и все наши рассуждения про их полноту остаются в силе.

4. ИСТОРИЧЕСКОЕ ОТСТУПЛЕНИЕ

Итак, мы обсудили три алгоритма решения проблемы начальника порта: алгоритм полного перебора, алгоритм эвристического перебора, геометрический алгоритм. По-видимому, читатели согласятся, что самый нетребовательный с информационной точки зрения и в тоже время самый изящный с интеллектуальной и эстетической точки зрения — это алгоритм эвристического перебора. Поэтому мы считаем уместным рассказать о контексте, в котором этот алгоритм был разработан Э. Дейкстрой.

Дело в том, что Э. Дейкстра рассматривал науку программирования не только как область приложения достижений других фундаментальных наук: математики, физики, лингвистики, психологии, и т.д., а как одну из фундаментальных наук наравне с перечисленными. Наука программирования имеет свой предмет исследований и методы. Разумеется, она может использовать методы и результаты других фундаментальных наук. Так, например, на наших глазах родились и развиваются такие направления науки программирования как квантовые, генетические и нейронные алгоритмы и вычислительные устройства. Но при этом Э. Дейкстра настаивал на том, что и наука программирования может обогатить другие фундаментальные науки идейно, а не только тем, что предоставит мощные вычислительные машины, средства и системы программирования. Может быть, прогресс биоинформатики — один из самых известных примеров такого благотворного идейного влияния науки программирования на биологию.

Пожалуй, наиболее яркий пример проникновения идей и методов чистой математики (и прежде всего алгебры, логики, теории множеств, топологии и геометрии) в науку программирования — это та ветвь науки программирования, которая получила название теоретического программирования. Она включает в себя такие направления, как семантику языков программирования, верификацию программ, теорию типов данных, анализ алгоритмов и т.п. К сожалению, известно мало примеров, когда идеи и методы из перечисленных отраслей теории программирования проникали в чистую математику и способствовали её прогрессу.

В то же время научное наследие Э. Дейкстры содержит множество примеров того, как идеи и методы теории программирования могут быть ис-

пользованы самым неожиданным образом в элементарной математике, комбинаторике, математическом анализе и т.д.[14] В частности, алгоритм, который мы здесь называем эвристическим перебором, был предложен Э. Дейкстрой как пример того, что может дать теория программирования для решения следующей неэлементарной задачи из элементарной планиметрии.

> Доказать, что любые заданные на плоскости N чёрных и N белых точек (среди которых нет коллинеарных) можно попарно соединить непересекающимися отрезками.

Первое, что сделал Э Дейкстра для решения этой планиметрической задачи, — это дал ей программистскую интерпретацию, а именно, сформулировал следующую алгоритмическую задачу.

> Спроектировать алгоритм, который по данным на плоскости N чёрным и N белым точкам (среди которых нет коллинеарных) строит N попарно непересекающихся отрезков, у каждого из которых один конец — это некоторая белая точка, а другой — это некоторая чёрная точка.

Дальнейшую историю читатель уже знает: для решения этой программистской задачи Э. Дейкстра применил программистский приём — выбрал эвристику и спроектировал свой эвристический алгоритм, а для доказательства тотальной завершаемости своего эвристического алгоритма применил метод потенциалов, принадлежащий другому классику науки программирования — Р. Флойду. После этого оставалось только сделать вывод, что исходная планиметрическая задача полностью решена в силу того, что эвристический алгоритм всегда успешно завершает свою работу.

Вообще алгоритмическая задача построения «хорошего» соединения для чёрных и белых точек на плоскости, к которой Э. Дейкстра «свёл» исходную планиметрическую задачу, известна в среде специалистов по анализу алгоритмов с конца 80-х годов прошлого века. Так, например, в популярном учебнике [1] есть задача 35-3 «Охотники за привидениями», в которой предлагается спроектировать алгоритм для N охотников с протонными пушками для охоты за N привидениями так, чтобы протонные пучки не

[14] Здесь уместно порекомендовать любознательным читателям обратиться к электронному архиву Э. Дейкстры на http://www.cs.utexas.edu/users/EWD/.

пересекались[15]. Указания к решению, сделанные в цитируемом учебнике, соответствуют нашему геометрическому методу. Однако в [1] предлагается использовать более тонкую геометрическую технику так называемых выпуклых оболочек для поиска прямой, которая отделяет равное число чёрных и белых точек. Такую прямую оказывается возможно найти за время пропорциональное $N \times \log N$ вместо нашей грубой оценки N^2 (желающим узнать, как именно это делать, предлагаем подумать, как оптимизировать наш грубый метод или рекомендуем обратиться к гл. 35 в [1]). В результате получается, что построить «хорошее» соединение чёрных и белых точек возможно за время пропорциональное $N^2 \times \log N$.

Однако даже приведённая оценка $N^2 \times \log N$ не является лучшей из известных для алгоритмической задачи о хороших соединениях чёрных и белых точек на плоскости. Дж. Хершберг и С. Сури в 1990 г. объявили, что данная алгоритмическая задача может быть решена за время пропорциональное $N \times \log N$ за счёт более быстрого построения выпуклой оболочки [2]. Но изложение соответствующей теории, алгоритмов и аспектов реализации уже выходит за рамки данной работы.

5. АНАЛИЗ АЛГОРИТМОВ

Разрешимая проблема — это задача, для решения которой в принципе существует алгоритм получения правильного ответа, причём этот алгоритм может не учитывать ограниченность продолжительности жизни человека, материальных ресурсов и т.д. Соответственно, **неразрешимая проблема** — это задача, для которой даже в принципе не существует алгоритма получения правильного ответа. **Теория алгоритмов** — это направление математики, которое занимается формализацией понятия алгоритм, для того что бы доказывать неразрешимость тех или иных проблем, а также изучает степени неразрешимости неразрешимых проблем[16]. **Анализ алгоритмов** — это направление теории программирования, которое занимается изучением того, как решать с учётом ограниченности продолжительности жизни человека, материальных ресурсов и т.д. с помощью ручки и бумаги, на реальных вычислительных машинах или на других «вычислительных устройст-

[15] На наш взгляд, формулировка задачи про охотников за приведениями довольно-таки искусственная по сравнению с формулировкой проблемы начальника порта.

[16] Обсуждение того, как доказывать неразрешимость какой-либо проблемы, и того, что неразрешимые проблемы могут быть по-разному или в разной степени неразрешимыми, выходит за пределы данной статьи.

вах» те проблемы, которые с точки зрения теории алгоритмов в принципе разрешимы, сравнивает эффективность различных алгоритмов и алгоритмическую сложность разрешимых проблем.

Проблема начальника порта в принципе разрешимая — ведь для нее мы уже подробно обсудили три различных алгоритма. Теперь необходимо детально проанализировать эти алгоритмы.

Для N чёрных и N белых точек существует N! взаимнооднозначных соединений. Теоретически не исключено, что алгоритму полного перебора придётся перебрать все N! соединений прежде, чем он найдёт «хорошее». Поэтому верхняя оценка числа повторений цикла WHILE в алгоритме полного перебора — N! К сожалению, эта верхняя граница является точной: для любого N можно привести пример, когда этот алгоритм выполнит ровно N! итераций цикла. Пусть (для определённости) чёрные точки — это B_1, … B_N, белые — это W_1, … W_N, ПЕРВ — это соединение $[B_1, W_1]$, … $[B_N, W_N]$, а СЛЕД перебирает перестановки белых точек в алфавитном порядке от $W_1, W_2, \ldots W_{N-1}, W_N$ до $W_N, W_{N-1}, \ldots W_2, W_1$. Тогда расположим чёрные и белые точки симметрично на верхней и нижней полуокружностях и перенумеруем их по часовой стрелке так, как это показано на рис. 4. Для таких точек существует единственное хорошее соединение $[B_1, W_N]$, $[B_2, W_{(N-1)}]$, … $[B_{(N-1)}, W_2]$, $[B_N, W_1]$. Но при переборе перестановок в алфавитном порядке это хорошее соединение будет получено в точности (N!)-м.

Для геометрического алгоритма мы уже объявляли кубическую верхнюю оценку сложности: число вызовов рекурсии ограничено сверху числом N, а число итераций цикла FOR на k-м вызове ($k \in [1..N]$) ограничено сверху $(N - k)^2$. Можно показать, что эти верхние оценки достаточно точны: для любого N можно привести пример, когда этот алгоритм выполнит ровно N вызовов рекурсии, а на k-м вызове ($k \in [0..N]$) он выполнит ровно $(N - k)$ итераций цикла FOR, т.е. всего этот цикл будет выполняться $N \times (N + 1)/2$ раз. Для определённости предположим, что чёрные точки — это B_1, … B_N, белые — это W_1, … W_N, а цикл FOR осуществляет перебор точек в порядке возрастания их номеров. Тогда в качестве примера рассмотрим уже знакомый нам пример на рис. 3: имеется N чёрных и N белых точек, они расположены в разных половинках окружности симметрично относительно диаметра. В таком случае на начальном (нулевом) вызове рекурсии цикл начнётся с $X = B_1$, перебёрёт все N точек $W_1, W_2, \ldots W_{N-1}, W_N$ в качестве значений для Y и остановиться с $X = B_1$ и $Y = W_N$. После этого произойдёт первый вызов рекурсии, и мы окажемся в ситуации, когда чёрных и белых точек осталось по $(N - 1)$, и они расположены в разных половинках окружности симметрично относительно диаметра. Следовательно (аналогично

нулевому вызову рекурсии), цикл будет исполнен ровно (N – 1) раз, после чего произойдёт второй вызов рекурсии и мы окажемся теперь в ситуации, когда чёрных и белых точек осталось по (N – 2), и они расположены в разных половинках окружности симметрично относительно диаметра. Продолжая этот процесс, мы получим ровно N вызовов рекурсии, причём, на k-м вызове (k ∈ [0..N]) он выполнит ровно (N – k) итераций цикла FOR, т.е. всего этот цикл будет выполняться N × (N + 1)/2 раз.

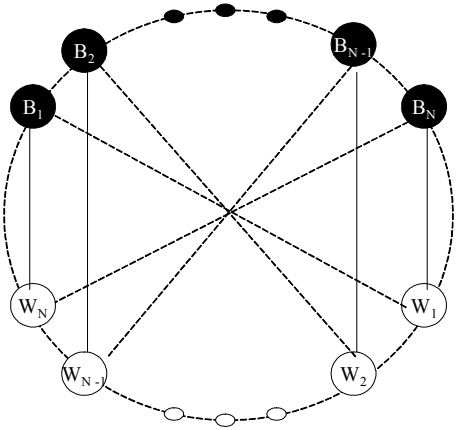

Рис. 4. Пример расположения чёрных и белых точек, при котором алгоритм полного перебора выполнит N! итераций цикла WHILE, начав с соединения, обозначенного пунктирными линиями, и закончив «хорошим» соединением, обозначенным сплошными линиями. На этом же примере геометрический алгоритм выполнит ровно N вызовов рекурсии, причём, на k-м вызове (k ∈ [1..N]) он выполнит ровно (N – k) итераций цикла FOR, т.е. всего этот цикл будет выполняться N × (N + 1)/2 раз

После такого успешного теоретического анализа алгоритма полного перебора и геометрического алгоритма, нас постигла неудача с теоретическим анализом алгоритма эвристического перебора: с одной стороны, этот алгоритм имеет довольно-таки очевидную верхнюю оценку сложности N! для числа итераций цикла WHILE, а с другой стороны, мы оказались не в со-

стоянии ни дать сколько-нибудь близкой нижней оценки числа итераций этого цикла в худшем случае, ни понизить верхнюю оценку. Поэтому мы, пользуясь случаем, формулируем в качестве открытой проблемы следующую задачу.

> Оценить сверху и снизу число итераций цикла в алгоритме эвристического перебора в худшем случае.

В сложившейся ситуации нам оставалось только одно средство для анализа алгоритма эвристического перебора: нам оставалось выполнить машинный эксперимент. Такой эксперимент был поставлен в следующих условиях. Для каждого $N \in [2..100]$ проводилось серия опытов (до 100). В каждом опыте N чёрных и N белых точек выбирались случайно (при равномерном распределении) в квадрате $[0..1] \times [0..1]$ (случаи коллинеарных точек отсеивались); для каждого опыта подсчитывались число «хороших» соединений, число итераций цикла WHILE в алгоритме полного перебора, число итераций цикла FOR в геометрическом алгоритме и число итераций цикла WHILE в эвристическом алгоритме, а также измерялось время работы соответствующих программ. Потом подсчитанные величины усреднялись по числу опытов в серии. Таким образом были получены семь эмпирических зависимостей[17] от числа $N \in [2..100]$:

- среднее число хороших соединений,
- среднее число итераций цикла WHILE в алгоритме полного перебора,
- среднее число итераций цикла FOR в геометрическом алгоритме,
- среднее число итераций цикла WHILE в эвристическом алгоритме,
- среднее время работы программы, реализующей алгоритм полного перебора,
- среднее время работы программы, реализующей геометрический алгоритм,
- среднее время работы программы, реализующей эвристический алгоритм.

В результате выполненных экспериментов оказалось, что:

- доля «хороших» соединений (т.е. среднее число хороших соединений делённое на N!) экспоненциально быстро убывает,

[17] Для числа хороших соединений и алгоритма полного перебора $N \in [2..13]$, что обусловлено большим временем работы программы.

– среднее число итераций цикла WHILE в алгоритме полного перебора пропорционально N!,
– среднее число итераций цикла FOR в геометрическом алгоритме изменяется по кубическому закону[18],
– среднее число итераций цикла WHILE в эвристическом алгоритме изменяется по квадратичному закону[19],
– программа, реализующая алгоритм полного перебора, проигрывает по производительности[20] программам, реализующим геометрический и эвристический алгоритмы;
– программы, реализующие геометрический и эвристический алгоритм, имеют примерно одинаковую производительность.

Результаты эксперимента можно суммировать следующим образом: лидером производительности оказались эвристический и геометрический алгоритмы и соответствующие им программы. Если успех геометрического алгоритма вполне предсказуем исходя из оценок его сложности в худшем случае, то для эвристического алгоритма мы не можем объяснить причины его успеха ссылкой на оценки сложности. Однако выполненный эксперимент позволяет нам сформулировать две гипотезы и предложить их в качестве проблем, открытых для исследования:

– доказать, что при равномерном распределении N чёрных и N белых точек (среди которых нет коллинеарных) в квадрате $[0..1] \times [0..1]$ доля соединений без пересечений среди всех возможных N! соединений стремиться к нулю с ростом N по экспоненциальному закону;
– формально оценить эффективность в среднем алгоритма эвристического перебора и сравнить её с эффективностью в среднем геометрического алгоритма для N чёрных и N белых точек (среди которых нет коллинеарных), равномерно распределённых в квадрате $[0..1] \times [0..1]$.

[18] $20N^3 - 210N^2 + 760N - 914$.
[19] $(N^2 - N + 2)/2$.
[20] Т.е. по среднему времени работы.

6. ПРОБЛЕМЫ ОЛИМПИАДНОГО ДВИЖЕНИЯ

После весьма подробного обсуждения моделей, алгоритмов и их программных реализаций для проблемы начальника порта мы можем вернуться к главной теме нашей статьи — обсуждению ситуации с олимпиадами по информатике и программированию.

Во-первых, не всякая задача по информатике обязательно заканчивается программной реализацией, есть проблемы, которые вообще не требуют компьютерного программирования. Примеров таких задач по информатике множество. Это могут быть

- задачи описания технологии использования имеющегося программного обеспечения для достижения определённой цели (например, технологии «программирования» кроссвордов в MS Excel);
- задачи выбора математической/информационной модели для неформально поставленной проблемы (например, проблема выбора системы записи арабских цифр почтовых индексов, узнаваемой человеком, но устойчивой к одиночным ошибкам);
- задачи выбора и высокоуровневого человеко-ориентированного дизайна (представления) алгоритма и структур данных для какой-либо проблемы в виде псевдокода и описания абстрактных типов данных (например, для проблемы начальника порта, как это было сделано выше для алгоритмов полного перебора, эвристического перебора и геометрического алгоритма);
- различные «обратные» задачи, когда по алгоритму и/или по математической модели требуется предложить неформализованную проблему, которая решается этим алгоритмом или формализуется в виде данной проблемы (например, для $D > 2$ описать неформальную проблему, которая бы могла быть формализована как D-мерная геометрическая задача о непересекающихся отрезках между белыми и чёрными точками).

Во-вторых, даже когда конечная цель задачи — работающая программа, автоматическое тестирование зачастую не в состоянии выявить лучшее решение, так как многие разумные критерии качества не сводятся к времени работы и времени разработки. Качественные критерии могут вообще не поддаваться объективным количественным оценкам, а по существу могут требовать участия экспертов с их субъективным мнением. Так, например, имеет смысл оценивать стиль программирования и соответствие программы высокоуровневому дизайну, контролировать обоснование корректности программы (т.е. доказательство и/или систему тестов), проверять её «отчу-

ждаемость» (т.е. комментированность и/или документированность) и, в конце концов, оценивать оригинальность принятых дизайнерских, алгоритмических и программистских решений. Всё это на данный момент совершенно не поддаётся автоматической оценке вообще, а уж тем более автоматическому тестированию и хронометражу.

Применительно к проблеме начальника порта, например, автоматическое тестирование в состоянии легко отсечь программные реализации алгоритмов полного перебора (или перебора с откатом). Наш эксперимент показал, что подобрать тесты, которые бы чётко разделяли программные реализации геометрического и эвристического метода — это сложная задача для составителей тестов. Но оригинальность эвристического алгоритма и простота реализации, на наш взгляд, заслуживают особой оценки, даже если на тестах и/или по хронометражу времени разработки его конкретная программная реализация проиграет программной реализации, например, геометрического алгоритма. Однако без доказательства корректности программу, реализующую алгоритм эвристического перебора, вообще не следует рассматривать, так как без этого доказательства эта программа — просто надежда на «авось»: авось эвристика да и сработает! Как следует из корректности алгоритма, подобрать тест, на котором эта эвристика бы не работала, невозможно; поэтому резонно потребовать у программиста доводов, почему эта эвристика надёжна. Участник олимпиады, спроектировавший, обосновавший и реализовавший эвристический алгоритм, демонстрирует умение оригинально мыслить и навыки, которые выходят за пределы «кулинарной книги» алгоритмов.

В заключение мы хотим сделать четыре основных вывода.

1. Мы вынуждены констатировать, что в последнее десятилетие наиболее популярные олимпиады по информатике трансформировались в олимпиады по программированию с автоматическим тестированием готовых программ в качестве мерила правильности. У этого явления были объективные и субъективные причины. Однако олимпиады по информатике (школьные, прежде всего) не должны сводиться к олимпиадам по программированию, а должны также содержать задачи на технологические навыки и знания, на умение строить модели, представлять информацию и алгоритмы в форме, ориентированной на человека и т.д.
2. Для того что бы подчеркнуть не только спортивный, но и профессиональный аспект олимпиад в области информатики, в части программистских задач нельзя ограничиваться автоматическим тестированием и хронометражем, а необходимо принимать во внимание

некоторые из критериев качества, такие как стиль программирования, обоснование корректности, комментированность и/или документированность, оригинальность принятых дизайнерских, алгоритмических и программистских решений. Возможно, даже имеет смысл организация олимпиад по информатике и программированию в два тура — бескомпьютерный и компьютерный[21].

3. Для того, чтобы подчеркнуть образовательный и исследовательский аспект олимпиад по информатике, в олимпиадах допустимо использовать задачи, которые выходят за пределы стандартных курсов по информатике (но не выходят за пределы общеобразовательного уровня участников), содержат элемент исследования (например, проверить, какой алгоритм будет работать быстрее в среднем) и подводят к открытой исследовательской проблеме. Такие задачи обязательно должны сопровождаться подробным разбором после олимпиады, причём на разборе следует обращать внимание на моменты, которые явно учат новому материалу или которые требуют дополнительных исследований.
4. К сожалению, труд организаторов олимпиад по информатике всё ещё не получил должной оценки. Это стало субъективной причиной подмены информатики программированием и предпочтения автоматического тестирования всестороннему анализу решений, в то время как информатика — это комплексная наука, включающая в себя и разработку моделей, и использование готовых технологий, и программирование, и разные методы исследований (от экспериментов до математических доказательств).

[21] Подобно международным олимпиадам по физике, которые включают теоретический и экспериментальный туры [3]: «The competition shall be conducted over two days, one for the theoretical examination and one for the experimental examination. There will be at least one full day of rest between the examinations. The theoretical examination shall consist of three theoretical problems and shall be of five hours total duration. The experimental examination shall consist of one or two problems and shall be of five hours total duration».

СПИСОК ЛИТЕРАТУРЫ

1. Корнмэн Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. — М: МЦНМО, 2000.
2. Hershberger J., Suri S. Applications of a semi-dynamic convex hull algorithm // Proc. of the 2nd Scandinavian Workshop on Algorithm Theory SWAT'90. — Lect. Not. Comp. Sci. — 1990. — Vol. 447. — P. 380–392.
3. Statutes of the International Physics Olympiads (version accepted in 1999 in Padova). At URL http://www.jyu.fi/tdk/kastdk/olympiads/.

Е. В. Бодин, Л. В. Городняя, Н. В. Шилов

ПО КАКОМУ ПРЕДМЕТУ ОЛИМПИАДА?

Препринт
126

Рукопись поступила в редакцию 20.10.05
Рецензент Т. Г. Чурина
Редактор З. В. Скок

Подписано в печать 01.11.05
Формат бумаги 60 × 84 1/16 Объем 1.7 уч.-изд.л., 1.9 п.л.
Тираж 100 экз.

ЗАО РИЦ «Прайс-курьер»
630090, г. Новосибирск, пр. Акад. Лаврентьева, 6, тел. (383) 330-72-02