Maciej Marek Sysło,
Anna Beata Kwiatkowska

ПРЕПОДАВАНИЕ МАТЕМАТИКИ С ОПОРОЙ НА КОМПЬЮТЕРНОЕ МЫШЛЕНИЕ

Цель вычислений – понимание, а не числа.
R.W. Hemming.

1. ВВЕДЕНИЕ

Эта статья является продолжением наших работ [25] и [28]. В [25] мы продемонстрировали, как обучение информатике может обогатить математическое образование в школе, а в [28] мы показали, как студенты высшей школы могут применять компьютерное мышление в своей работе, используя учебник информатики, построенный на проектном подходе к обучению. Мы, специалисты по информатике, твердо убеждены, что компьютерное мышление полезно в математике и в других науках. Хотя все вопросы, рассмотренные в [25], включены в учебник информатики [9], они все еще отсутствуют в учебниках математики.

Очень важным дополнительным преимуществом является возможность конструирования содержательных математических объектов в компьютерной среде. Таким образом развивается компьютерное мышление, полезное при решении математических задач, а также и в других дисциплинах.

Maurer показал [16], что «алгоритмическая математика имеет два абсолютно разных значения»: традиционное и современное, причем «под традиционной алгоритмической математикой понимается выполнение готовых алгоритмов, в то время как современная занимается их созданием и рассчитана на мышление в терминах алгоритмов, предназначенных для решения задач и развития теории». В настоящее время, 30 лет спустя после [16], алгоритмы используются в преподавании математики все еще в традиционном стиле. Мы надеемся, что подходы, применяемые в данной статье и продемонстрированные на нескольких примерах, изменят роль вычислений в преподавании математики. Алгоритмика в ее современном значении использовалась в книгах [22, 23], в которых студенты изучали алгоритмы, развивая их.

2. КОМПЬЮТЕРНОЕ МЫШЛЕНИЕ

Jeannette Wing [31] утверждала, что компьютерное мышление – важнейшее свойство, необходимое для существования в современном мире. Его важность не ограничивается областью вычислений, оно является базой при рассмотрении различных задач и поиске их решений.

Компьютерное мышление в информатике традиционно рассматривалось как алгоритмическое мышление, цель которого – сформулировать решение задачи в виде ал-

горитма, преобразующего вход в выход, и записать этот алгоритм в виде компьютерной программы. Компьютерное мышление как более общий термин включает в себя мышление на разных уровнях абстракции и является присущим информатике подходом к решению задач, который позволяет всем ученикам применять компьютеры и вычислительные навыки к различным объектам в науке и приложениях.

Существует несколько описаний термина компьютерное мышление, однако нет точного его определения. Denning отметил в [8], что компьютерное мышление – не единственная адекватная характеристика информатики, и он прав – хотя компьютерное мышление возникло из вычислений, но применимо ко всем областям человеческой деятельности. Ясно, что знание основ информатики помогает систематично, эффективно обрабатывать информацию и решать задачи. Lu и Fletcher обсуждали в [15] роль программирования в изучении информатики и сформулировали тезис: «программирование для информатики – то же самое, что построение доказательства для математики». Вообще, не только в информатике программирование – средство общения с компьютером с помощью языка программирования. Чтобы получить решение задачи на компьютере, нужно сначала найти решение «теоретически», а затем сообщить это решение компьютеру с помощью программы. Для этого существуют различные пути (см. [26]). В настоящее время реализуются несколько инициатив, таких как The Khan Academy [13], The Hour of Code [10] и Scratch learning community [20]. Более того, существует заметное движение за раннее привлечение учеников к программированию (см. [6]). Мы планируем осуществить это в ближайшем будущем в Польше – включить программирование в различные школьные предметы.

Несмотря на свое происхождение из информатики, компьютерное мышление не сводится к изучению компьютеров и компьютерных программ, хотя они играют важную роль в построении решений задач. Компьютерное мышление – способ обдумывания задач в компьютерных терминах. Предполагается, что ученики приобретают умение так моделировать задачу, чтобы получить компьютерное решение; они учатся разделять результат решения и процесс, с помощью которого этот результат получен. Весьма важно, что компьютерное мышление полезно почти во всех школьных предметах, так как оно дает понимание того, что может быть или не может быть вычислено. Оно поощряет студентов к созданию компьютерных моделей задач, которые вроде бы никак не связаны с вычислениями. Компьютерное мышление включает в себя понятия, навыки и компетенции, составляющие саму суть вычислений, такие как абстракция, декомпозиция, обобщение, аппроксимация, эвристика, построение алгоритмов, вопросы эффективности и сложности.

Абстракция и декомпозиция используются в компьютерном мышлении при решении сложных задач, например, содержащих большие объемы данных. Задача переформулируется так, что она может быть эффективно решена путем трансформации, редукции и моделирования.

Компьютерное мышление включает такие характерные для информатики инструменты, как рекурсивное и алгоритмическое мышление (см. [30, 29]).

Мощным инструментом являются эвристические рассуждения, которые позволяют находить решения даже при недостаточных знаниях ученика и тем самым развивают его творческое мышление.

Понятие сложности играет важную роль в компьютерном мышлении. Понятие эффективности с точки зрения затрат времени и сложности алгоритма используется при планировании решения и последующей оценки результатов. Мощность современных суперкомпьютеров недостаточна для решения некоторых задач реального мира. С другой стороны, неразрешимость некоторых задач используется в криптографии для обеспечения безопасности данных.

Стоит заметить, что применение ИКТ (информационных и коммуникационных технологий) не ограничивается созданием про-

граммных инструментов. Напомним различие между информатикой (созданием программ, компьютеров, теорий и т. д.) и ИКТ (применение инструментов информатики) [26]. Создание инструментов (то есть программ) и получение новой информации требует использования абстракций, манипулирования данными и многих других концепций и идей информатики, как это здесь описывается на примере школьной математики.

Сейчас общепринято следующее определение компьютерного мышления [33]: компьютерное мышление – это мыслительный процесс, с помощью которого задачи и их решения формулируются и представляются в форме, пригодной для их эффективного решения посредством совместной работы человека и компьютера.

Мы предлагаем при решении задач применять компьютерное мышление в рамках схемы, основанной на определении компьютерного мышления, данном в ISTE [12] и CSTA [7] и похожем на описание стратегии решения задач из [26]. Компьютерное мышление можно описать как процесс, состоящий из следующих шагов:

- Формулировка задачи в виде, пригодном для применения компьютера для ее решения.
- Организация и анализ данных.
- Представление данных с помощью абстрактных моделей.
- Построение решения на основе алгоритмического подхода.
- Реализация и анализ решения с целью отыскания наиболее эффективного по затратам времени ресурсов решения.
- Обобщение процесса решения и перенос его на широкий круг задач.

3. КОНСТРУКТИВНОЕ ОБУЧЕНИЕ МАТЕМАТИКЕ

В этом разделе мы рассматриваем несколько тем, которые обычно входят в курс информатики в школах, и показываем, как их можно использовать в преподавании математики. Все эти темы включены в школьные учебники информатики [7] в Польше. Однако большинство из них отсутствуют в наших учебниках математики.

3.1. ПРЕДСТАВЛЕНИЕ ЧИСЕЛ

Компьютерное мышление предполагает выбор подходящего представления чисел [31] в решаемой задаче. Существуют два основных представления чисел – точный (тип integer в языках программирования) и приближенный (тип real). Результаты действий с числами типа integer также относятся к типу integer, а результаты действий с числами типа real всегда имеют тип real, то есть являются приближенными. Приближенные вычисления мы рассмотрим в пункте 3.3, а здесь мы вкратце обсудим представление чисел типа integer.

Ученики обычно знакомы с алгоритмом получения двоичного представления неотрицательного числа n, основанном на делении числа на 2 и последовательном получении остатков в качестве двоичных цифр. *Однако они затрудняются определить сколько битов* займет число n в памяти компьютера? Ответ на этот вопрос весьма важен, так как он связан с решением многих теоретических и алгоритмических задач. В [29] мы объясняем его связь с логарифмом и даем алгоритмическое определение логарифма:

Логарифм $\log_2 n$ равен числу шагов при последовательных делениях n на 2. (*От ред.:* верно для степеней двойки, в общем случае нужно вычислять целую часть логарифма $[\log_2(n-1)] + 1$).

В [29] мы демонстрируем, насколько важна такая интерпретация логарифма для создания и оценки решений многих задач, таких как поиск в упорядоченных множествах, вычисление значений экспоненты, отыскание наибольшего общего делителя двух чисел и применение стратегий «разделяй и властвуй».

Двоичное представление целых чисел и вообще представление чисел в позиционной системе с произвольной базой дает возможность рассматривать в школе многочлены высоких степеней, в отличие от теперешней программы, в которой присутствуют многочлены только первой и второй степени. Представление числа n в системе с основанием p выражается многочленом от переменной p с коэффициентами из множества «цифр»

{0, 1, …, p – 1}. При таком представлении действия с целыми числами могут производиться с помощью алгоритмов, созданных для действий с многочленами. Например, в правиле Горнера [25] быстрый алгоритм вычисления экспоненты был создан на основе двоичного представления экспоненты в виде многочлена.

Арифметика сравнений по модулю используется в криптографии. Чтобы объяснить ученикам алгоритм RSA [9], мы вводим «арифметику часов» (*от ред.* модульную арифметику) – ученикам понятна арифметика по модулю 24, которую мы используем в повседневной жизни.

3.2. РЕДУКЦИЯ И ДЕКОМПОЗИЦИЯ

Редукция – это подход к решению задач, а *редуктивное мышление* – это определенный способ мышления. Решение задач с помощью редукции является важным методом в преподавании вообще и особенно в преподавании математики и информатики. Редукция – один из навыков, характеризующих компьютерное мышление [31].

В алгоритмике редукция в ее простейшей форме может быть описана следующим образом: выяви связи между решаемой задачей (P) и задачей (S) , которую мы знаем, как решать, преобразуй вход в задачу P во вход в S, а затем выход задачи S в выход P. В более общем случае задача P может быть разбита на несколько задач S. Кроме того, в качестве задачи S можно взять ту же задачу P, но с некоторым подмножеством входных данных, и тогда получаем рекурсивный алгоритм (см. п. 3.4.). В теоретической информатике концепция редукции, известная как полиномиальная эквивалентность задач по времени, используется в курсах по теории вычислимости и сложности.

Редуктивная стратегия тесно связана с абстракцией – во-первых, выявления связи между двумя задачами P и S требует игнорирования их несущественных характеристик и сосредоточения на их сходстве и, во-вторых, детали решения S несущественны для решения P. Тогда при создании задачи P нам достаточно знать, что решение задачи S существует и может быть эффективно получено. Остальные детали решения игнорируются – работа ведется на более высоком уровне абстракции.

В остальной части этого пункта мы показываем на трех задачах, как с помощью редукции можно найти их решения. Во всех этих задачах в качестве задачи S используется одна и та же задача отыскания наибольшего общего делителя нескольких чисел.

Сведение исходной задачи к такой же задаче, но с некоторым подмножеством исходных данных, производится с помощью рекурсивных процедур, подробности о рекурсивном мышлении в пункте 3.4.

Отыскание наибольшего элемента в множестве

Отыскание наименьшего или наибольшего (или наилучшего) элемента в множестве элементов данного типа – один из наиболее часто встречающихся видов деятельности в повседневной жизни. Поэтому представляет интерес поиск наиболее эффективного способа решения этой задачи.

Решая такие задачи, мы обычно абстрагируемся от типа элементов и рассматриваем их как числа, а в качестве базовой операции используем сравнение. Обычно мы перебираем подряд все элементы, сохраняя текущий наибольший элемент. Следовательно, имея n элементов, мы должны сделать (n – 1) сравнение, иногда при этом обновляя текущий элемент. Такой поиск называется *линейным*.

На возражение некоторых учеников, что в турнирах число сравнений (партий) меньше, чем (n – 1), мы просим их провести турниры с разным числом участников. В результате они обнаруживают, что число сравнений такое же, как и при линейном поиске. На этом этапе урока мы пытаемся убедить учеников, что (n – 1) – это наименьшее число сравнений, необходимое для отыскания наибольшего (наилучшего) элемента среди n элементов. Для этой цели мы используем очень простой и красивый аргумент, который приводит Hugo Steinhaus в своей прекрасной книге [21]: Если Джон выиграл тур-

нир, то каждый из остальных участников проиграл по крайней мере $(n-1)$ раз (Джону или другим игрокам). Значит, было сыграно не меньше, чем $(n-1)$ партий. Итак, линейный (или) турнирный алгоритм оптимальны, так как они производят точно $(n-1)$ сравнений.

Далее мы спрашиваем учеников, является ли проигравший в финальной партии вторым по силе игроком турнира? Ответ на этот вопрос не очень очевиден.

Одновременное отыскание наименьшего и наибольшего элементов

Задачу можно решить, отыскав сначала наименьший элемент, затем – наибольший среди остальных элементов. Такой алгоритм требует $(2n-3)$ сравнения.

Однако при таком методе оба элемента ищутся не одновременно. Чтобы построить алгоритм одновременного поиска, мы задаем ученикам следующий вопрос: когда мы ищем наименьший и наибольший элементы, какой вывод следует из неравенства $x < y$, которому удовлетворяют два числа из данного множества? Ясно, что x – кандидат на наименьший элемент, а y – кандидат на наибольший элемент. Поэтому разбиваем множество на два подмножества одинакового размера[1] – подмножество кандидатов на наименьший элемент и подмножество кандидатов на наибольший элемент. Так задача сводится к двум подзадачам: найти наименьший элемент среди подмножества кандидатов на наименьший элемент и найти наибольший элемент среди подмножества кандидатов на наибольший элемент. В этом алгоритме разбиение требует $n/2$ сравнений, а две подзадачи требуют $(n/2-1)$ сравнений каждая. В итоге сложность алгоритма равна $(3n/2-2)$. Если n нечетное, то сложность равна $[3n/2-2]$. Интересно, что этот алгоритм тоже оптимален, если оптимальны алгоритмы для подзадач.

Сортировка

Сортировка встречается во многих реальных ситуациях, и ее можно рассматривать с разных точек зрения: практического применения, алгоритмической техники, вопросов сложности. Если ученики знают, как найти наименьший элемент в данном множестве, они легко приходят к идее о том, как свести сортировку множества из n элементов к повторному применению процедуры поиска наименьшего элемента к последовательности подмножеств k неупорядоченных элементов, где $k = n, n-1, \ldots, 2$. На каждом шаге выполняется $k-1$ сравнение для поиска наименьшего элемента, поэтому сложность всей сортировки равна $n(n-1)/2$. При изучении сложности этого алгоритма отметим два момента. Во-первых, студенты могут с помощью некоторых программных пакетов [24] проверить, что сложность не зависит от начального порядка в данном множестве, даже если оно изначально упорядочено. Во-вторых, существуют алгоритмы, более быстрые и практически и теоретически. Студенты должны понять, что применение оптимального алгоритма к подзадачам не всегда приводит к оптимальному алгоритму для исходной задачи.

Треугольники

Следующая задача была использована в [25] для иллюстрации различия между двумя ее решениями. Одно решение было дано на уроке математики, а другое – во время урока информатики. Пусть множество A содержит большое количество целых чисел, например 10 миллионов. Требуется проверить, может ли каждая тройка чисел из множества A служить длинами сторон треугольника. Очевидно, что нужно проверить, что любые a, b, c удовлетворяют условиям для сторон треугольника.

«Математическое» решение. Проверяем выполнение условий для сторон треугольника для всевозможных троек чисел из A. Если множество A содержит n целых чисел, то производим Cn^3 действий (сложения и сравнений), где C постоянная. При большом n алгоритм очень непрактичен.

Решение, использующее редукцию. Здесь мы настраиваем учеников на получение практичного компьютерного решения. Во-первых, они должны заметить (и дока-

[1] Для простоты полагаем, что множество содержит четное число элементов.

зать), что если $a \le b \le c$, то тройка a, b, c удовлетворяет требуемым условиям тогда и только тогда, когда $a + b > c$. Отсюда ученики легко получают, что если все тройки чисел из A должны удовлетворять этим условиям, то нам достаточно проверить выполнение условий для двух наименьших чисел и самого большого числа в A. После короткого обсуждения ученики убеждаются, что нам не нужно сортировать все множество A, чтобы найти эти три числа – их отыскание сводится к одному прогону линейного поиска в A. Сложность этого решения пропорциональна числу элементов в A.

Этот пример показывает, что поиск компьютерного решения математической задачи приводит к элегантному и эффективному решению, причем не только с компьютерной, но и с математической точки зрения.

3.3. АППРОКСИМАЦИЯ

Аппроксимацию можно рассматривать с разных сторон и в математике и в информатике. В математике она является задачей, главным образом, теоретического анализа, а в информатике существуют по меньшей мере три причины для ее изучения:

– приближенное представление чисел влияет на точность вычислений и в некоторых случаях приводит к распространению ошибок округления;

– компьютер выполняет только 4 основных арифметических действия, поэтому остальные действия (извлечение корня, вычисление тригонометрических функций и т. п.) выполняются с помощью последовательности вычислений значений многочленов, и, значит, важную роль играет аппроксимация функций многочленами;

– существует много очень важных с практической точки зрения, но не поддающихся теоретическому исследованию задач оптимизации, для которых только приближенные решения реальны, так как точные решения недоступны для компьютеров.

В этом пункте мы рассмотрим задачи первых двух категорий, а в пункте 3.5 обсудим эвристический подход к задачам третьей из указанных выше категорий, который позволяет находить приближенные решения.

Ошибки округления

Решение квадратного уравнения

$$ax^2 + bx + c = 0$$

одна из стандартных тем школьной математики. Обычно, применяется D-алгоритм, где $D = b^2 - 4ac$, и если $D > 0$, то корни вычисляются по формулам:

$$x_1 = (-b - \sqrt{D})/(2a) \text{ и}$$
$$x_2 = (-b + \sqrt{D})/(2a).$$

Однако, когда b и $\sqrt{D}$ близки друг к другу (то есть у них совпадают значащие цифры), то один из корней (он близок к 0) теряет значащие цифры. В этом случае предлагается одна из формул Виета: $x_1 x_2 = c/a$. Предположим, что b и $\sqrt{D}$ близки друг к другу, тогда если b и $\sqrt{D}$ разных знаков, то вычисляем x_2 по приведенной выше формуле, а $x_1 = c/(a\, x_2)$, иначе – вычисляем x_1 по приведенной выше формуле, а $x_2 = c/(a\, x_1)$. Нам не встречался школьный учебник математики, в котором квадратное уравнение решалось бы таким «компьютерным» способом.

Вычисление квадратного корня

Почти во всех практических задачах используются вещественные числа (нецелые), и решения обычно получаются приближенными. Метод получения приближенного решения, как правило, состоит из последовательности шагов, приближающих к точному решению. В школьной математике извлечение квадратного корня используется для демонстрации такого метода.

Пусть $x = \sqrt{a}$, где $a > 0$. Исходя из начального приближения x_0, последовательные приближения $x_1, x_2, \ldots$ вычисляются по формуле: $x_i = (x_{i-1} + a/x_{i-1})/2$ и, к нашему удивлению, школьные учебники математики (не только в Польше) приводят эту формулу как «черный ящик», не объясняя ее происхождение. Однако мы можем смоделировать вычисление, используя очень простой геометрический аргумент, понятный школьнику, который знает только формулы площадей квадрата и прямоугольника и что такое среднее двух чисел. Очевидно, если $x = \sqrt{a}$, то $x^2 = a$. Значит, найти x означает найти длину

стороны квадрата, площадь которого равна a. Если наше предположение относительно величины x неверно и мы знаем, что площадь равна a, то другая сторона прямоугольника равна a/x. Если значение x слишком мало, то значение a/x слишком велико, а если x слишком велико, то значение a/x будет слишком мало. Отсюда школьник легко сделает вывод, что правильный ответ находится где-то между этими двумя числами, и в качестве следующего приближения мы можем принять среднее арифметическое этих двух чисел. Отсюда и получаем нашу формулу.

Обобщая эту идею и взяв куб размерности k (квадрат – это двумерный куб), ученики получают формулу для вычисления корня порядка k.

Вычисления типа итераций могут производить в Excel (см. [25]) даже школьники. Они могут экспериментировать с различными начальными значениями и убедиться, что небольшое число итераций приводит к хорошему приближению.

В заключение отметим, что геометрическое моделирование и графические инструменты могут помочь в развитии и представлении вычислений без привлечения таких более продвинутых средств, как дифференциальное исчисление. С другой стороны, эти методы можно использовать для введения более сложных разделов. Другой аспект вычисления площади является предметом образовательных программ [24].

3.4. РЕКУРСИЯ

Рекурсия – это не отдельная тема, а скорее метод и инструмент, способ мышления, используемый для декомпозиции (редукции) задачи на подзадачи того же типа. Для решения задачи таким методом нужно определить декомпозицию задачи на подзадачи и композицию решения исходной задачи из решений подзадач. В некоторых случаях рекурсия используется для сведения задачи к той же задаче, но с меньшими значениями параметров, например, факториал, бинарный поиск, алгоритм Евклида. Рекурсию можно рассматривать как альтернативу итерациям (например, вычисление чисел Фибоначчи), но для нас интерес к ней основан на том, что она является мощным средством построения решений и выполнения их на компьютере.

Рекурсия особенно популярна при решении задач на компьютере, так как в рекурсивных программах значительная часть работы осуществляется самим компьютером и не должна быть явно выписана в программе. Рекурсивные алгоритмы и программы обычно короче и более «читабельны», чем их итеративные аналоги, хотя и выполняются гораздо дольше.

Существует несколько типов рекурсий: линейная (например, факториал), множественная (например, числа Фибоначчи, бинарное дерево), вложенная (функция Аккермана) и взаимная (система двух рекурсий, в которой рекурсивные вызовы одной рекурсии обращены к другой рекурсии). Множественную рекурсию называют также экспоненциальной, так как ее сложность не полиномиального типа.

Рекурсия все еще представляет трудность для учителей и учеников – для них это одна из самых трудных тем в дискретной математике. В работе [30] мы представляем различные стороны рекурсии, а также средства и примеры, которые могут помочь учителям объяснить, а ученикам понять, что такое рекурсивное мышление. Существует несколько методов и инструментов для введения и объяснения рекурсии: индукция, стек, трассировка и рекурсивное дерево. Эти методы чаще всего используются для визуализации рекурсии и объяснения структуры рекурсивных вызовов. Учитывая трудности введения, объяснения и использования рекурсии, мы отдельно описываем наши подходы, инструменты и методы. Рекурсию можно вводить как «сюжет из реальной жизни», и тогда программа, визуализирующая рекурсивные вычисления, может помочь в преодолении трудностей, возникающих при первом знакомстве с рекурсией. Мы также советуем, как наиболее эффективно использовать рекурсию.

Существует тесная связь между двумя понятиями: (математическим) индукцией и (компьютерным) рекурсией, хотя решения

задач с помощью рекурсии более популярны в информатике, чем в математике, по крайней мере, в школе. Авторы [15] заявляют, что «рекурсия – это исполнимая версия индукции». На вопрос, какая из двух тем проще, с какой нужно начинать обучение, – они отвечают: «рекурсия более доступна, чем индукция… сама по себе рекурсия не проще, чем индукция. Скорее, ее использование на компьютере проще индуктивных доказательств с карандашом и бумагой». Наш педагогический опыт подтверждает этот вывод.

Рекурсия и рекурсивное мышление, по-видимому, являются наиболее надежным связующим звеном между преподаванием математики и преподаванием информатики. Мы убеждены, что введение рекурсии на уроках информатики вносит свой вклад и в математическое образование и в решение чисто математических задач.

3.5. ЭВРИСТИЧЕСКОЕ МЫШЛЕНИЕ

Эвристика – основанная на эксперименте техника решения задач, применяемая, когда другие методы слишком медленны, или для отыскания приближенного решения, когда не удается найти точное решение. При этом жертвуют оптимальностью и точностью ради скорости. Цель эвристики – получить решение за разумное время, причем достаточно хорошее в терминах поставленной задачи. Эвристические решения могут не быть наилучшими и обычно являются только приближенными, но они приемлемы, так как их отыскание не требует много времени. Эвристические методы – единственные практические методы для решения таких сложных задач оптимизации, которые, с одной стороны, не поддаются теоретическому исследованию (NP-трудные), а, с другой – их необходимо решать в реальных ситуациях. Для таких задач исчерпывающее исследование непрактично, и эвристические методы позволяют найти приемлемое решение.

Джордж Пойя (Дьёрдь Полия), написавший книгу «Как решать задачу» (1945 г.), признан отцом эвристического мышления в математике.

Наиболее общий эвристический метод – это метод проб и ошибок, применимый почти к любой задаче в математике и других дисциплинах. В информатике и математике очень популярен, особенно в задачах оптимизации, эвристический метод, называемый жадным алгоритмом.

Этот алгоритм делает локально оптимальный выбор на каждом шаге с надеждой, что эта стратегия приведет к глобальному оптимуму. Для многих задач это неверно, но такой метод позволяет получить приближенное решение за разумное время. В некоторых случаях можно оценить разницу между полученным и оптимальным решениями.

Далее мы кратко опишем две теоретически очень сложные задачи (подробнее в [23, 26]), но жадный алгоритм позволяет получить для них довольно хорошие решения.

Жадная стратегия является очень естественной при решении задач поиска оптимального пути, например, когда мы хотим достигнуть определенного пункта кратчайшим путем или за кратчайшее время при выполнении некоторых дополнительных условий для сети и пути. В простейшем варианте этой задачи greedy алгоритм дает оптимальное решение (смотри ниже), но в общем случае это не так, например при поиске кратчайшего замкнутого пути, проходящего через каждый город один раз.

Стоит отметить, что за последние 20–25 было предложено несколько эвристик, называемых метаэвристиками, которые основаны на естественных, заимствованных из природы и науки принципах, например поиск с запретами (табуированный поиск), алгоритм имитации отжига, генетический алгоритм, муравьиный алгоритм. Теоретическое обоснование этих эвристик отсутствует, однако на практике они дают неплохие решения, отличающиеся на 2–5 % от оптимальных.

Задача о размене

Следующая задача предлагается школьникам: для данной суммы денег V найти наименьшее число банкнот и монет, эквивалентное данной сумме V. Предполагается,

что все банкноты и монеты имеются в неограниченном количестве. Сначала, не прибегая к абстракции, ученики обсуждают, как он получают сдачу в магазине – они часто получают много мелких монет. Обсуждение приводит к лучшему пониманию задачи и обычно заканчивается выработкой жадной стратегии: на каждом шаге выбирай наибольшую возможную банкноту или монету. Чтобы поэкспериментировать с этой стратегией для различных значений V, мы просим учеников построить (закодировать) их решение (жадный алгоритм) на рабочем листе Excel и протестировать решения при разных значениях V. Таким образом, задача становится задачей программирования для Excel.

Затем мы обычно ставим несколько дополнительных задач, например: действительно ли их программа находит наименьшее число монет и банкнот для любого значения V? Большинство учеников отвечает ДА, но не может объяснить это друг другу и учителю. Чтобы показать, что жадный алгоритм не всегда приводит к оптимальному решению, мы просим учеников добавить к нашей системе (она состоит из монет: 1, 2, 5, 10, 20, 50, 100, 200, 500 грошей и нескольких монет) новую монету ценой в 21 грош и применить жадный алгоритм для $V = 63$ гроша. Вместо оптимального решения из трех монет (21 + 21 + 21) жадный алгоритм находит 4 монеты (50 + 10 + 2 + 1). Довольно часто случается, что подходящих монет просто нет. Мы просим учеников найти такие подмножества монет и значений V, для которых жадный алгоритм не гарантирует оптимального решения, и, более того, для некоторых значений V невозможно подобрать подходящий набор монет.

Задача о размене оказалась очень интересной для учеников, они активно обсуждали возможные решения для всех вариантов задачи, как запрограммировать жадный алгоритм и протестировать его при разных значениях V.

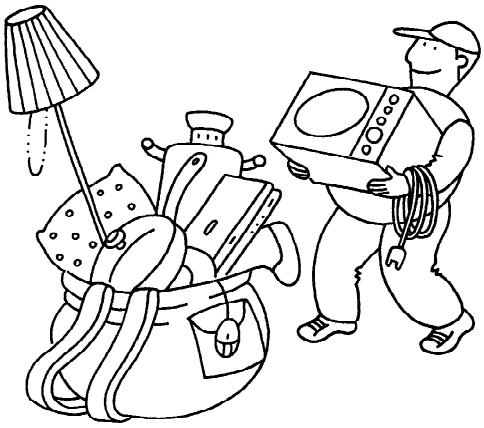
Вообще для решения задачи о размене требуется динамическое или целочисленное программирование. Интересно, что для большинства валютных систем, включая

польский злотый, евро и доллар США жадная стратегия действительно находит оптимальное решение.

Задача о рюкзаке

Укладка рюкзака – еще одна задача, решаемая с помощью жадного алгоритма: в рюкзак заданной емкости нужно упаковать несколько предметов, каждый из которых имеет определенный вес и цену. Цель – уложить в рюкзак предметы максимального общего веса W и максимальной цены.

Ученики обычно принимают одну из трех стратегий: в первую очередь класть самый дорогой предмет, класть самый легкий предмет или класть предмет с максимальным отношением цены к весу. Затем они усиленно пытаются найти пример, в котором ни одна из этих стратегий не приводит к оптимальному решению. Мы поощря-



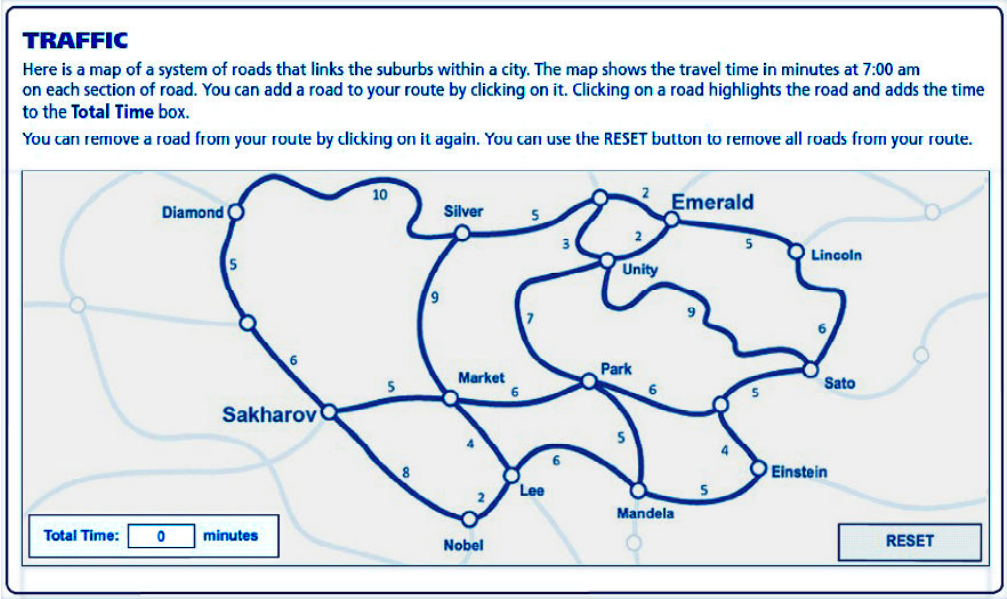


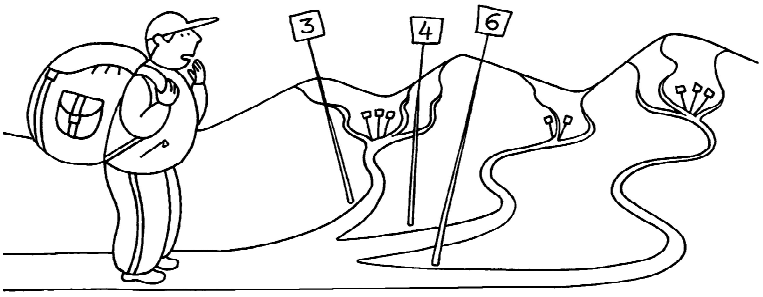
Рис. 1. Отыскание кратчайшего пути от Sakharov до Emerald (PISA 2012 Test, OECD [18])

ем учеников применять для этой задачи алгоритм динамического программирования (см. [22, 23]), который можно рассматривать как рекуррентное соотношение, основанное на принципе оптимизации Беллмана и примененное к двумерному массиву.

Задача о кратчайшем пути

Стратегия, основанная на жадном алгоритме и примененная к некоторым задачам оптимизации, порождает оптимальные решения, причем это довольно просто доказывается. Задача о кратчайшем пути представляет семейство таких задач. Рассматривается сеть, состоящая из точек (например, городов или пересечений дорог) и дорог между этими точками. Каждой дороге между двумя точками приписана определенная длина (или время, требуемое на ее прохождение). В простейшем варианте задача заключается в том, чтобы найти кратчайший путь, соединяющий две заданные точки и состоящий из дорог этой сети. Такая задача обычно иллюстрируется рисунками типа рис. 1 и 2.

Сначала естественно попробовать решить задачу с помощью жадного алгоритма. Студенты начинают применять алгоритм, называемый алгоритмом «ближайшего соседа» и определяемый следующим образом: начиная с исходной точки, на каждом шаге, кроме последнего, выбирается ближайшая точка. Сетка дорог на рис. 1 дает контрпример – кратчайший путь имеет длину 20 и не может быть получен с помощью алгоритма «ближайшего соседа».



Слегка модифицированный жадный алгоритм приводит к оптимальному решению данной задачи. Этот алгоритм (алгоритм Дейкстры) определяется так: на каждом шаге выбирай точку, не совпадающую с уже пройденными точками и ближайшую к начальной точке.

Другая задача, связанная с задачей о кратчайшем пути, была в конкурсе Bebras 2013 для школьников. Задача состояла в определении числа кратчайших путей между двумя данными точками на сетке бобровых дорожек с препятствиями (обозначенными X) (см. рис. 2).

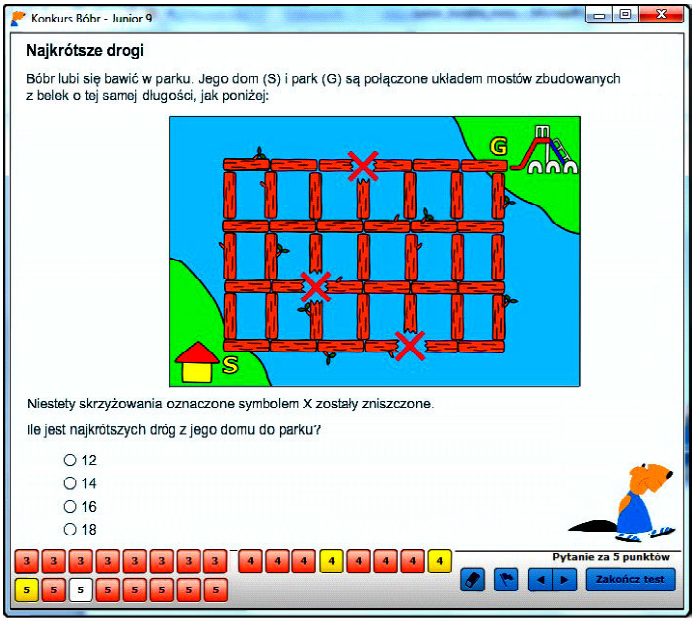


Рис 2. Подсчет числа кратчайших путей между *S* и *G* (конкурс Bebras, 2013 [4])

4. ВЫВОДЫ

В данной статье мы рассматривали проблему применения компьютерного мышления в школьной математике. Преследовалась двойная цель:

1) расширить и обогатить традиционные темы школьной математики, получая с помощью компьютерного мышления решения, использующие методы информатики как дисциплины и компьютеры как инструмент вычислений;

2) внести вклад в конструктивное обучение математике, обучение посредством создания объектов реального мира.

Вопросы, рассмотренные в данной статье, связаны с следующими понятиями: представление данных (типы integer и real), редуктивное мышление (сведение сложных задач к простым), приближенное решение вычислительных и сложных задач, вопросы сложности (рекурсивное и алгоритмическое мышление) и эвристика (задача размена, укладка рюкзака, задача о кратчайшем пути).

Литература

1. ACM K-12 Task Force Curriculum Committee. A Model Curriculum for K-12 Computer Science, ACM, 2003.
2. *Armoni M., Gal-Ezer J., Tirosh D.* Solving problems reductively // Educational Computing Research 32(2), 2005. P. 113–129.
3. *Armoni M.* Reductive thinking in a quantitative perspective: the case of the algorithm course // ITiCSE. Madrid (Spain), 2008. P. 53–57.
4. Bebras Competition: http://www.bebras.org/.
5. Computational thinking / http://www.iste.org/standards/computationalthinking.aspx.
6. Computer science: A curriculum for schools, March 2012 / http://www.computingatschool.org.uk/data/uploads/ComputingCurric.pdf.
7. CSTA: Computational Thinking Task Force / http://csta.acm.org/Curriculum/sub/CompThinking.html
8. *Denning P.J.* The profession of IT beyond computational thinking // Comm. ACM 52(6), 2009. P. 28–30.
9. *Gurbiel E., Hard-Olejniczak G., Koiczyk E., Krupicka H., Syslo M.M.* Informatics (In Polish), Vols. 1 and 2. Textbook for high school, WSiP, Warszawa (2002–2003).
10. Hour of Code: http://csedweek.org/.
11. *Hu C.* Computational thinking – what it might mean and what we might do about it // ITiCSE, Darmstadt (Germany), 2011. P. 223–227.
12. ISTE: http://www.iste.org/learn/computational-thinking.
13. Khan Academy: https://www.khanacademy.org/.
14. *Leron U., Zaskis R.* Computational recursion and mathematical induction // For the Learning of Mathematics 6(2), 1986. P. 25–28.

15. *Lu J.J., Fletcher G.H.L.* Thinking about computational thinking, *SIGCSE*, Chattanooga (USA), 2009. P. 260–264.

16. *Maurer S.B.* Two meanings of algorithmic mathematics // The Mathematics Teacher 77(1984). P. 430–435.

17. *Papert S., Harel I.* Situating constructionism // Constructionism, Ablex, 1991.

18. PISA 2012 Results (OECD): http://www.oecd.org/pisa/keyfindings/pisa-2012-results.htm

19. *Polya G.* How to solve it. Princeton University Press, 1957.

20. Scratch: http://scratch.mit.edu/.

21. *Steinhaus H.* Mathematical Snapshots, G.E. Stechert Press (1939); 3rd edition, Oxford University, Press, New York, 1969.

22. *Sysło M.M.* Algorithms (in Polish), WSiP, Warszawa 1997.

23. *Sysło M.M.* Pyramids, Cones and Other Algorithmic Constructions (in Polish), WSiP, Warszawa, 1998.

24. *Sysło M.M.*: educational software: http://mmsyslo.pl/Materialy/Oprogramowanie.

25. *Sysło M.M., Kwiatkowska A.B.* Contribution of Informatics Education to Mathematics Education in Schools, in: Mittermeir R.T. (ed.) // ISSEP 2006, LNCS 4226, Springer-Verlag, Berlin, Heidelberg, 2006. P. 209–219.

26. *Sysło M.M., Kwiatkowska A.B.* The challenging face of informatics education in Poland, in: Mittermeir R.T., Sysło M.M. (eds.) // ISSEP 2008, LNCS 5090, Springer-Verlag, Berlin, Heidelberg 2008. P. 1–18.

27. *Sysło M.M.* Outreach to Prospective Informatics Students, in: Kalaš I., Mittermeir R.T. (eds.) // ISSEP 2011, LNCS 7013, Springer-Verlag, Berlin, Heidelberg 2011. P. 56–70.

28. *Sysło M.M., Kwiatkowska A.B.* Informatics for All High School Students, A Computational Thinking Approach, in: Diethelm I., Mittermeir R.T. (eds.) // ISSEP 2013, LNCS 7780, Springer-Verlag, Berlin, Heidelberg, 2013. P. 43–56.

29. *Sysło M.M., Kwiatkowska A.B.* Think logarithmically!, accepted for KEYCIT, Potsdam (Germany), July 2014.

30. *Sysło M.M., Kwiatkowska A.B.* Introducing students to recursion: a multi-facet and multi-tool approach, accepted for ISSEP 2014, Istanbul (Turkey), September 2014.

31. *Wing, J.M.* Computational thinking // Comm. ACM 49(3), 2006. P. 33–35.

32. *Wing J.M.* Computational thinking and thinking about computing //Phil. Trans. R. Soc. A 366(2008). P. 3717–3725.

33. *Wing J.M.* Research notebook: computational thinking – what and why? / http://link.cs.cmu.edu/article.php?a=600.

Maciej Marek Sysło,
Faculty of Mathematics and Informatics, Nicolaus Copernicus University, Poland,
Faculty of Mathematics and Informatics Institute of Computer Science, University of Wroclaw,

Anna Beata Kwiatkowska,
Faculty of Mathematics and Informatics, Nicolaus Copernicus University, Poland.

Перевод М.И. Юдовина.