# Master thesis preamble

Dmitry Anisimov

December 25, 2017

Below, you can find my master thesis defended at the faculty of Applied Mathematics and Control Processes of the Saint-Petersburg State University in Saint-Petersburg, Russian Federation. This thesis is written in Russian language and is the result of my literature search in the topic of DNA computers to show my ability to comprehend complex mathematical subjects at the end of my master program.

During the fifth year of the university, I became interested in DNA computers and how different properties of DNA molecules can be used for solving various computationally hard real-life problems. Since this was not a very popular topic at that time, my goal was to spread it over the Russian-speaking scientific community. And so, this thesis does not bring any new significant results but rather is the short summary of what was known at the moment of its publication in the corresponding field with some applications to solving NP-hard problems. The only intention of this summary is to elaborate-popularize on the topic of DNA computing and show its importance for the future of the computational mathematics. I also implemented a small tool in C sharp language to help newcomers to comprehend this rather difficult topic a bit faster.

This work is mostly based on the literature indicated at the end of the thesis and, if you want to know more about the topic, this is the place where you need to look at and these are the authors of all main results discussed in this thesis. Some pictures used in this thesis are also due to the authors of the original results or are taken from the related open sources. The format of the thesis conforms to the writing format accepted at the year of the thesis's publication at the faculty of Applied Mathematics and Control Processes of the Saint-Petersburg State University.

Санкт-Петербургский Государственный Университет

Факультет Прикладной Математики – Процессов Управления

Кафедра Математической Теории Моделирования Микропроцессорных Систем Управления

Анисимов

Дмитрий

Владимирович

# Решение NP-полных задач с использованием свойств ДНК.

Заведующий кафедрой: доктор технических наук

профессор **Меньшиков Г.Г.**

Научный руководитель: доктор физ.-мат. наук

профессор **Братчиков И.Л.**

Рецензент:

Санкт-Петербург

2010

**Содержание:**

# Введение.

Настоящая работа посвящена вопросам применения новой парадигмы вычислений на основе ДНК для решения ряда известных задач, не поддающихся решению стандартными методами. А именно мы остановимся на так называемом классе NP-полных задач.

Всем хорошо известно, что в современном мире большинство математических задач, полностью или частично, решается с помощью компьютеров. Почему же ЭВМ, разработанная уже более 50 лет назад, столь прочно удерживает свои позиции в этой области? Во-первых, это связано со скоростью обработки информации: с помощью компьютера можно решить большинство насущных проблем за доли секунд. Во-вторых, компьютер доступен почти каждому и не требует большого объема знаний для решений определенного круга задач. Требуется, всего лишь найти алгоритм решения и запрограммировать его на любом из известных языков программирования, воспользоваться одним из мощных математических пакетов, таким как Maple, MathCAD, MatLab и т.д., или просто найти готовое решение в интернете. Люди же занимающиеся наукой по настоящему используют его не только для решения задач, но и для визуализации своих идей, постановки экспериментов, вычисления вероятностей тех или иных событий и т.д.

С момента появления первого компьютера прошло уже много времени, он очень сильно изменился и по виду и по размеру и по скорости обработки информации. Те задачи, которые нельзя было решить еще 10 лет назад из-за его малой скорости, легко решаются сейчас. Таким образом, одной из главных задач человечества в нынешний информационный век является увеличение скорости обработки информации, для того чтобы решать все более и более трудоемкие задачи. Решение же этой проблемы тесно связано с другими не менее важными проблемами, такими как поиск новых, более эффективных алгоритмов вычислений, минитюаризация вычислительных устройств и их взаимодействие друг с другом. Так раньше, компьютеры весили несколько тон, могли вычислять лишь элементарные задачи, и для этого требовалось проводить серию трудоемких операций, а сейчас, некоторые из них не весят и килограмма, молниеносно обрабатывают большие объемы информации и имеют дружелюбный интерфейс.

Таким образом, решено уже множество трудоемких задач, о вычислении которых еще пару десятков лет назад не могло идти и речи. Но и осталось множество задач, которые человек, даже с помощью такого мощного инструмента как компьютер, решить не может. Какие-то задачи лишь частично, а какие-то и полностью. К этому классу задач как раз таки и относится класс NP. А относится он следующим образом: это класс задач, для которых не найдены полиномиальные алгоритмы (то есть алгоритмы, работающие за практически приемлемое время), но и не доказано, что их не существует. Грубо говоря, для них существуют алгоритмы решения, то есть эти задачи имеют конечное решение, но они не эффективны, даже с учетом использования такого быстрого устройства, как компьютер. Следовательно требуется либо доказать, что таких эффективных алгоритмов не существует, либо найти альтернативные пути их решения. В данной работе мы как раз таки и займемся вторым вариантом, то есть рассмотрим новый путь решения этого класса задач.

Как видно из названия работы, в качестве нового способа решения этих проблем мы выбрали молекулы ДНК (дезоксирибонуклеиновые кислоты) и алгоритмы, основанные на этих молекулах. Это связано с тем, что в последнее время все большее внимание уделяется алгоритмам, основанным на естественных способах вычислений, то есть на том, как вычисляет природа. И это не просто так. Многие опыты и эксперименты, проведенные человеком, показали, что природные механизмы решения самые эффективные и быстрые,

поэтому разумнее воспользоваться этими готовыми решениями, нежели придумывать некие искусственные способы. Люди все лучше и лучше осознают, что ничего лучше, чем создала природа, пока что они придумать не могут. Взять хотя бы, например, нас – людей, разве хоть один современный робот может сравниться с человеком. Поэтому просто необходимо разобраться в том, как устроены природные механизмы обработки информации. И чем больше мы в этом разбираемся, тем больше понимаем, что многие задачи, казавшиеся неприступными раньше, легко решаются сегодня, например, с помощью тех же ДНК. Но здесь, естественно, возникают и свои специфические проблемы, о которых также будет упомянуто в данной работе.

Также сделаем акцент на том, что даже сейчас разработано мало программного обеспечения, основанного на природных алгоритмах или их моделирующих. Хотя такое программное обеспечение очень важно, т.к. оно упрощает и удешевляет проведение экспериментов, изучение новой отрасли и позволяет не тратить впустую время и деньги на то, чего можно было и не делать. Например, программный комплекс Xgrow имитирует процесс синтеза различных структур и оценивает возможные ошибки при создании структуры. А система Namot представляет собой удобное графическое средство для работы с молекулярными структурами, позволяющее составлять структуры из атомов, создавать связи в трехмерном пространстве и строить последовательности молекулярных операций.

Здесь мы упомянули массовый перебор. Стоит заметить, что именно он лежит в основе молекулярных алгоритмов, описанных в данной работе. Но его успешное применение в общем случае возможно лишь в устройствах, сильно отличающихся от компьютера. Сейчас эти устройства называют ДНК-компьютерами. С историей их появления вы можете ознакомиться в следующем разделе.

В конце данной работы приводится пример комплекса, разработанный специально для обучения пользователей новым методам вычислений, основанных на ДНК.

Теперь приведем краткое содержание основной части данной работы:

Часть 1 содержит 2 главы, а именно:

Глава 1 включает краткое описание строения ДНК и основы для создания нового типа вычислительных устройств, т.е. так называемых ДНК-компьютеров. Стоит отметить, что ДНК-компьютер сильно отличается от того, что мы привыкли понимать под обычным компьютером. Также в этой главе приводятся различные модели молекулярных вычислений, такие как модель параллельной фильтрации, стикерная и плиточная модели, а также модель вычислений в терминах теории формальных языков.

Глава 2 включает в себя краткое описание проблемы P = NP и понятия класса NP-полных задач. В ней же приводятся оценки сложности некоторых из этих задач а также примеры их решений в терминах молекулярных вычислений.

Часть 2 содержит 2 пункта, а именно:

Первый пункт содержит описание новой системы DNA-Learning, обучающей пользователей новым алгоритмам и моделям вычислимости, основанных на ДНК. Второй пункт содержит примеры использования этой системы.

# История.

В данном разделе проведем небольшой экскурс в историю зарождения ДНК-компьютеров - приспособлений, предназначенных для решения трудоемких задач, чтобы лучше понять смысл появления этой области в современной науке.

“Компьютеры будущего смогут весить не более чем 1,5 тонны”. Так говорили популярные механики в 1949 году. Большинству из нас сегодня - в век современных технологий, миниатюрных компьютеров и сотовых телефонов это утверждение может показаться смешным. Так как с тех дней мы сделали колоссальный прорыв в миниатюризации: от компьютеров, занимающих целые залы до современных настольных ноутбуков. Идея же живых ячеек и молекулярных комплексов, рассмотренная, как замена механическим и электронным устройствам, датирована концом 1950 годов, когда Ричард Фейнман изложил свою известную работу, описывающую построение субмикроскопических компьютеров. Но даже сейчас до субмикроскопического уровня очень далеко. Современные электронные устройства с каждым годом становятся все меньше и меньше, но постепенно люди подходят к пределу их миниатюризации и это вынуждает нас переходить на молекулярный уровень. Но здесь, как и везде, требуется прочная математическая основа. Чем недавно впрочем, и стали заниматься ученые. Они вернулись к этой области науки, и развивают вычисления, основанные на массовом переборе всех возможных решений, поставленной задачи, используя методики молекулярной биологии. Отметим, что некоторые современные задачи можно решить только массовым перебором их значений, с чем обычный компьютер справится, просто, не может в силу своей архитектуры. А если некоторые из этих задач он и решит, то на это будет затрачено столько времени (вплоть до 30 лет), что об эффективности и разумности таких методов говорить уже не приходится, в отличие от ДНК-компьютеров, легко воспринимающих такой способ вычислений. Первое же появление ДНК-компьютера датируется концом 1994 года, когда Л. Эдлман, профессор университета Южной Калифорнии объявил, что он решил небольшой пример трудоемкой задачи о гамильтоновом пути для данного ориентированного графа, используя всего одну маленькую пробирку ДНК. Представляя информацию в виде последовательности ДНК-молекул, Эдлман наглядно продемонстрировал использование существующих биотехнологических методик по манипулированию ДНК, как простой инструмент вычислений, основанный на массовом переборе.

Немного подробнее про опыт Эдлмана:

Как сказано выше Л. Эдлман решал задачу Коммивояжера. Суть ее в том, чтобы найти маршрут движения с заданными точками старта и финиша между несколькими городами (в данном случае семь), в каждом из которых можно побывать только один раз. Эта задача решается прямым перебором, однако при увеличении числа городов сложность ее возрастает. Классические компьютерные архитектуры требуют множества вычислений с опробованием каждого варианта.

Метод ДНК же позволяет сразу сгенерировать все возможные варианты решений с помощью известных биохимических реакций. Затем возможно быстро отфильтровать именно ту молекулу-нить, в которой закодирован нужный ответ.

*Проблемы, возникающие при этом:*

1. *Требуется чрезвычайно трудоёмкая серия реакций, проводимых под тщательным наблюдением.*
2. *Существует проблема масштабирования задачи.*

Биокомпьютер Эдлмана отыскивал оптимальный маршрут обхода для 7 вершин графа. Но чем больше вершин графа, тем больше биокомпьютеру требуется ДНК-материала. Было подсчитано, что при масштабировании методики Эдлмана для решения задачи обхода не 7 пунктов, а около 200, вес ДНК для представления всех возможных решений превысит вес нашей планеты.

Следующим, разработанным уже в 2001 году ДНК-компьютером, был однозадачный биокомпьютер Шапиро. Эхуд Шапиро реализовал модель биокомпьютера, который состоял из молекул ДНК, РНК и специальных ферментов. Для работы биокомпьютера необходимо составить правильную молекулярную смесь. Приблизительно через час смесь самостоятельно порождает молекулу ДНК, в которой закодирован ответ на поставленную перед вычислителем несложную задачу. В этом биокомпьютере ввод и вывод информации, а также роль программного обеспечения берут на себя молекулы ДНК. В качестве же аппаратного обеспечения выступают два белка-энзима естественного происхождения, которые манипулируют нитями ДНК. При совместном замешивании молекулы программного и аппаратного обеспечения гармонично воздействуют на молекулы ввода, в результате чего образуются выходные молекулы с ответом. В одной пробирке помещается около триллиона элементарных вычислительных модулей. В результате скорость вычислений достигает миллиарда операций в секунду, а точность 99,8 %. Биокомпьютер Шапиро мог применяться лишь для решения самых простых задач, выдавая всего два типа ответов: истина или ложь. Функционирование ДНК-компьютера сходно с функционированием теоретического устройства, известного в математике как «конечный автомат» или машина Тьюринга.

За время, прошедшее с момента выхода в свет ДНК-компьютеров, описанных выше, достигнут значительный прогресс в развитии, как теории, так и практики молекулярных вычислений. Современные ДНК-компьютеры, сконструированные в последние годы, существенно отличаются от громоздких «пробирочных» моделей, используемых в опытах выше. В этих компьютерах весь процесс фильтрации происходит в тонкой стеклянной трубке (длина 35 см, внутренний диаметр 0,3 см), заполненной гелем, и осуществляется с помощью гель-электрофореза. В гель заранее встроены участки, к которым прикреплены молекулы-зонды, соответствующие элементам рассматриваемой задачи, например, дизъюнктам исследуемой пропозициональной формулы, которую мы проверяем на выполнимость или общезначимость. (Здесь ярко проявляется, как прогресс биохимических технологий – в данном случае, недавно разработанный метод закрепленных цепочек ДНК на гелеобразующем полимере – воздействует на практику молекулярных вычислений.) Цепочки ДНК, кодирующие интерпретации переменных, перемещаются от одного такого участка к следующему под действием тока. «Гелевый» ДНК-компьютер компактен и допускает автоматическое управление процессом фильтрации. Еще более существенным его преимуществом является то, что фильтрация происходит быстрее и с гораздо более высокой точностью.

Если рассматривать, в общем, современные направления вычислительных процессов, то нам открываются все новые и новые тайны природы, позволяющие создавать устройства, вычисляющие биологическими методами и сильно отличающиеся от привычной в нашем понимании ЭВМ. Например, в самое последнее время внимание информатиков привлекают вычислительные процессы в живых клетках. Так, с вычислительной точки зрения чрезвычайно интересна процедура сборки генов при половом размножении одноклеточных класса ресничных (к нему принадлежит знакомая по школьным урокам зоологии инфузория-туфелька). Удивительно, но хранение генетической информации в так называемом генеративном ядре инфузории организовано по тому же принципу, по которому размещаются файлы на жестоком диске современного компьютера! При этом сложный процесс извлечения фрагментов гена из ДНК ядра с

последующим соединением их в нужном порядке происходит на несколько порядков быстрее, точнее и энергетически выгоднее, чем гораздо более примитивные операции в опытах по ДНК-вычислениям. Такие факты окончательно убеждают, что путь к созданию по-настоящему эффективных ДНК-компьютеров лежит через осознание того, как вычисляет мать-природа.

# Часть 1. Решение NP-полных задач средствами ДНК.

## Глава 1. Принцип молекулярных вычислений.

### • Основы для создания ДНК-компьютеров и строение ДНК.

Как мы уже отмечали выше, у современных компьютеров есть свои «камни преткновения». Существует множество непреодолимых для них задач, которые требуют массового перебора всех значений. Здесь к нам на помощь и приходят ДНК-компьютеры и различные молекулярные модели вычислений.

Чарльз Беббидж, знаменитый предтеча компьютерной эры, в 1810-1820 гг. спроектировал автоматический калькулятор, «Разностную машину», а затем и более амбициозную «Аналитическую машину». Главным препятствием для реализации этих проектов стало отсутствие аккуратно обработанных деталей, а также механических и электрических устройств, появившихся только в XX веке. Возможно, сегодня мы находимся в сходной ситуации по отношению к ДНК-компьютеру. Биохимической технике пока недостает точности, и направление ее развития не вполне адекватно специфическим запросам молекулярных вычислений, но весьма вероятно, что здесь период ожидания будет намного короче, чем в случае Беббиджа.

Надежды на успешное будущее молекулярных вычислений и ДНК-компьютеров основаны на двух фундаментальных феноменах. Это:

*(а) Массированный параллелизм цепочек ДНК*
*(б) Комплементарность Уотсона-Крика.*

Вкратце обсудим роль этих феноменов.

*(а)* Большинство знаменитых «труднорешаемых» задач, как уже упоминалось, может быть решено путем исчерпывающего перебора всех допустимых вариантов. Непреодолимая пока трудность кроется в том, что такой перебор слишком объемен, чтобы его можно было осуществить, используя существующие технологии. Однако благодаря плотности, с которой можно хранить информацию в цепочках ДНК, и простоте, с которой можно размножать эти цепочки, исчерпывающий перебор становится реальным. Типичным примером может служить задача раскрытия шифра: в ней можно одновременно испробовать все ключи.

*(б)* Данное свойство заключается в том, что когда (в идеальных условиях) возникает связь между двумя цепочками ДНК, то мы знаем, что противоположные друг другу основания комплементарны. Итак, если известна одна компонента получившейся молекулы, то известна и другая, и это не нуждается в какой-либо дополнительной проверке. Это обстоятельство приводит к мощному инструменту вычислений. Различным образом, кодируя информацию в цепочках ДНК, подлежащих связыванию, мы способны делать далеко идущие выводы, основываясь лишь на факте, что связывание имело место.

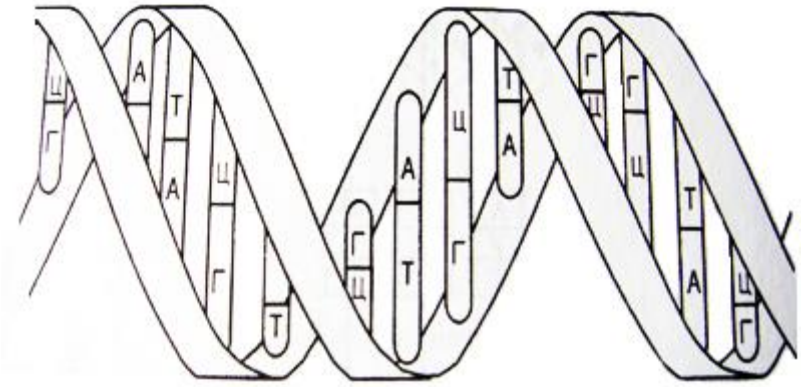

Рис. 1. Двойная спираль ДНК

Рассмотрим парадигму комплементарности а также строение молекул ДНК чуть подробней. ДНК состоит из полимерных нитей, обычно именуемых цепочками. Эти цепочки составлены из нуклеотидов, которые различаются только своими основаниями. Таких оснований четыре: А (аденин), Г (гуанин), Ц (цитозин) и Т (тимин). Хрестоматийная двойная спираль ДНК возникает при связывании двух отдельных цепочек. При образовании двойных цепочек и проявляется феномен, известный как комплементарность Уотсона-Крика. Связывание происходит за счет попарного притяжения оснований: А всегда связывается с Т, а Г с Ц. Комплементарность при образовании двойных цепочек схематически представлена на рис. 2.

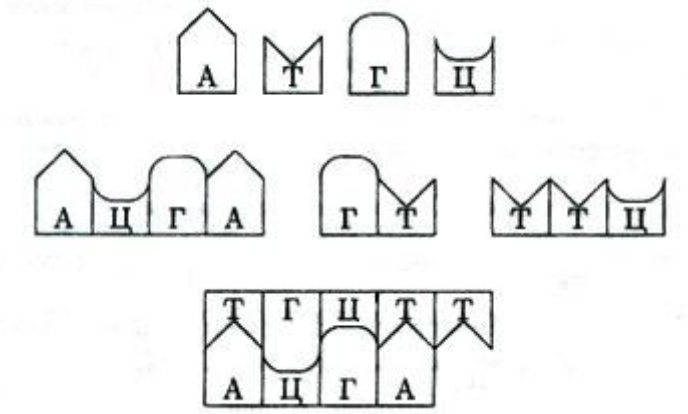


Рис. 2 Комплементарность Уотсона-Крика.

Как уже было сказано, нуклеотиды образуют полимерные нити. Иногда такие полимеры достигают в длину десятки миллионов оснований. В этих супермолекулах фосфат и дезоксирибоза играют роль поддерживающей структуры (они чередуются в цепочке), а азотистые соединения кодируют информацию. Молекула получается направленной: начинается с фосфатной группы и заканчивается дезоксирибозой. Длинные цепочки ДНК называют нитями, короткие - олигонуклеотидами.

Т.о. двойная спираль, обеспечивает возможность удвоения ДНК при размножении клетки. Задача удвоения решается с помощью специального белка-энзимы - полимеразы. Синтез начинается, только если к ДНК прикреплён кусочек её дополнения. Данное свойство активно используется в молекулярной биологии и молекулярных вычислениях. По сути своей полимераза - это реализация машины Тьюринга, состоящая из двух лент и программируемого пульта управления. Пульт считывает данные с одной ленты, обрабатывает их по некоторому алгоритму и записывает на другую ленту. Полимераза также последовательно считывает исходные данные с одной ленты (ДНК) и на их основе формирует ленту с результатом вычислений (дополнение Уотсона - Крика). Наличие

ошибок при операциях с цепочками ДНК может привести к весьма серьезным погрешностям. Поэтому, в конечном счете, успех идеи молекулярных вычислений, как уже было отмечено выше, существенно зависит от развития адекватной лабораторной техники.

## • Модели молекулярных вычислений.

В данном разделе мы попытаемся рассмотреть наиболее распространенные и испытанные модели молекулярных вычислений и/или так называемые молекулярные алгоритмы. В данный момент уже разработано большое количество таких моделей и алгоритмов, отличающихся между собой по биохимическим основам, по архитектуре, по набору допустимых операций и т.п. Многие из них даже весьма остроумны, но о степени их практической значимости судить пока трудно. Начнем мы с наиболее распространенной и впервые появившейся в опыте Эдлмана модели параллельной фильтрации, а затем рассмотрим стикерную модель, разработанную американским ученым С. Ровейсом. Обе модели заключаются в построении последовательности действий с пробирками, содержащими молекулы ДНК по некоторому алгоритму, разрабатывающемуся в соответствии с решаемой задачей, с целью получения желаемого результата, т.е. решения поставленной задачи. Ключевым элементом каждой из них является пробирка. Также и в одной и в другой модели срабатывает парадигма комплементарности Уотсона - Крика в плане построения двойных цепочек. Только в каждой из этих моделей они строятся по-своему. Таким образом, обе модели в чем-то похожи, а в чем-то имеют и сильные отличия. Затем приведем пример еще одной интересной модели молекулярных вычислений, основанной на представлении молекул ДНК в виде двумерных плиток, впервые появившейся в опытах Э. Винфри. Ну и в конце рассмотрим одну из самых интересных моделей, позволяющую довольно полно моделировать ДНК-вычисления на обычном компьютере, так называемую модель вычислений в терминах теории формальных языков, разработанную Г. Пауном, Г. Розенбергом и А. Саломаа.

### A. Модель параллельной фильтрации

В данной модели основной упор делается на фильтрацию потому, что множество всевозможных решений задачи получается уже на первом шаге за счет того, что взаимодействующие молекулы ДНК спроектированы нужным образом. А основная часть алгоритма – это извлечение нужного результата из множества всевозможных результатов.

Решение задачи начинают с множества коротких одинарных цепочек, которые соединяют шаг за шагом так, что после каждого шага остаются липкие концы. Получившиеся при этом одинарные цепочки кодируют все заданные входные данные. Затем происходит реакция сшивки, то есть все комплементарные пары находят друг друга и образуют большое количество двойных цепочек, которые не должны иметь одноцепочечных промежутков. В результате чего образуется множество всевозможных решений задачи. После чего идет ряд операций, представляющих собой фильтрацию, то есть отсеивание ненужных результатов.

Вычисление в этой модели есть последовательность операций слить, размножить, разделить и обнаружить. Входом и выходом вычисления являются пробирки. Чтение выхода заключается в определении последовательности олигонуклеотидов, кодирующих результат. Далее более подробно рассмотрим эти операции и их использование, но для начала введем понятие пробирки.

Пробирка – это мультимножество слов (конечных строк) над алфавитом {А, Ц, Г, Т}. (С интуитивной точки зрения пробирка – это вместилище одинарных цепочек ДНК. Цепочки присутствуют в пробирке с некоторыми кратностями, т.е. там может содержаться несколько копий одной и той же цепочки).

Начальная пробирка создается способом, указанным в начале этого пункта и содержит коды всех возможных решений задачи.

Следующие основные операции были первоначально определены для пробирок, т.е. мультимножеств одинарных цепочек ДНК.

*Слить.* Образовать объединение N1 ∪ N2 (в смысле мультимножеств) двух данных пробирок N1 и N2.

*Размножить.* Изготовить две копии данной пробирки N. (Заметим, что такая операция имеет смысл только для мультимножеств.)

*Обнаружить.* Возвратить значение истина, если данная пробирка N содержит, по крайней мере, одну цепочку ДНК, в противном случае возвратить значение ложь.

*Разделить* (или *Извлечь*). По данным пробирке N и слову w над алфавитом {А, Ц, Г, Т} изготовить две пробирки +(N,w) и –(N,w) такие, что +(N,w) состоит из всех цепочек в N, содержащих w в качестве (последовательной) подстроки, а –(N,w) состоит из всех цепочек в N, которые не содержат w в качестве подстроки. Четыре операции слить, размножить, обнаружить и разделить позволяют составлять программы, отвечающие на простые вопросы о наличие или отсутствии определенных подслов.

**Примеры:**

1) *Данная программа распознает, содержит ли данная пробирка цепочку, в которой встречаются оба пурина А и Г.*

**(1) Ввести (N)** // вводим начальную пробирку
**(2) N ← +(N, А)** // выделяем пробирку, вкл. все цепочки, сод. в качестве последовательной подстроки элемент Аденин
**(3) N ← +(N, Г)** // из полученной на пред. шаге пробирки выделяем пробирку, вкл. все цепочки, сод. в качестве последовательной подстроки элемент Гуанин
**(4) Обнаружить (N)** // выделяем полученную цепочку, если такая существует

2) *Эта программа извлекает из данной пробирки все цепочки, содержащие по крайней мере один из пуринов А и Г, сохраняя кратности этих цепочек.*

**(1) Ввести (N)** // вводим начальную пробирку
**(2) Размножить (N) на N1 и N2** // делаем две копии этой пробирки
**(3) Nа ← +(N1, А)** // выделяем из первой копии пробирку, вкл. все цепочки, сод. в качестве последовательной подстроки элемент Аденин
**(4) Nг ← +(N2, Г)** // выделяем из второй копии пробирку, вкл. все цепочки, сод. в качестве последовательной подстроки элемент Гуанин
**(5) –Nг ← –(Nг, А)** // выделяем из пробирки получ. на пред. шаге пробирку, вкл. все цепочки, не сод. в качестве последовательной подстроки элемент Аденин, получили цепочки только с Гуанином, т.е. избавляемся от повторений
**(6) Слить (Nа, –Nг)** // Объединяем полученные цепочки

Кроме четырех основных операций, перечисленных выше, можно ввести еще несколько операций типа *Разделить*:

*Разделить по длине.* По данным пробирке N и целому числу n, изготовить пробирку (N, ≤n), состоящую из всех цепочек из N длины не больше n.

*Разделить по префиксу (суффиксу).* По данным пробирке N и слову w, изготовить пробирку B(N, w) (соответственно E(N, w)), состоящую из всех цепочек в N, начало (соответственно конец) которых совпадает со словом w.

Таким образом, введя такой набор операций, мы получили простое средство для написания программ различного типа, решающих задачи в рамках модели параллельной фильтрации. О действительной реализуемости данных операций мы здесь не говорим, так как такая реализуемость в большей степени зависит не от последовательности действий, описанных в программе, а от микробиологических технологий, реализующих эти операции. Для простоты будем называть описанное нами средство языком программирования. Соответственно аналогично обычным языкам программирования здесь могут использоваться циклы для многоразового повторения некоторых операций, вида

**For (int i=0, i≤значение, i++) {операция};**

А также нетрудно показать, что все эти операции легко запрограммировать на обычном компьютере в виде аналогичных функций. Следовательно, в какой-то степени эти операции универсальны в плане места их использования. Но универсальность операций не исключает того свойства, что реальные задачи больших размерностей на обычном компьютере можно только моделировать, а решение возможно лишь на ДНК-компьютере.

Вычислительная парадигма, ассоциированная с моделью параллельной фильтрации, состоит в решении трудных задач путем полного перебора всех молекул ДНК, находящихся в начальной пробирке и кодирующих всевозможные решения задачи, с целью нахождения верного решения.

### B. Стикерная модель.

Стикерная модель базируется на парадигме комплементарности Уотсона-Крика, благодаря которой цепочки ДНК можно использовать как физический носитель информации. По существу, стикерная модель есть память с произвольным доступом, причем не требующая удлинения цепочек и допускающая, по крайней мере, теоретически, многократное использование.

Прежде всего, опишем основанный на комплементарности метод представления информации цепочками ДНК. В этом методе используется два основных типа одноцепочечных молекул ДНК: запоминающие цепочки и цепочки-наклейки или стикеры. Запоминающая цепочка длиной n нуклеотидов содержит k непересекающихся подцепочек, каждая из которых имеет длину m нуклеотидов. Таким образом, должно выполняться неравенство n≥mk.

| | | | |
|---|---|---|---|
| ГАГАГ | ТТТТЦ | АААААА | ЦЦЦЦЦ |

| |
|---|
| ТТТТТ |

| |
|---|
| ЦТЦТЦ |

| |
|---|
| ГГГГГ |

| |
|---|
| АААГ |

Рис. 3. Пример стикерной памяти.

В ходе вычисления каждая подцепочка представляет одну фиксированную булеву переменную (или, что эквивалентно, один бит). Подцепочки должны существенно отличаться одна от другой: в любых двух из них, по крайней мере, на нескольких позициях должны стоять разные нуклеотиды. Это нужно для того, чтобы каждый бит мог быть с уверенностью идентифицирован. Каждый стикер имеет длину m нуклеотидов и комплементарен в точности одной из k подцепочек запоминающей цепочки.

Каждая подцепочка запоминающей цепочки может быть либо включена, либо выключена. Говорят, что подцепочка включена, если к ней присоединен соответствующий ей стикер. В противном случае, если к подцепочке не присоединен стикер, говорят, что она выключена. Термин запоминающий комплекс будет использоваться для запоминающих цепочек, в которых включены некоторые подцепочки. Итак, запоминающие комплексы – это частично сдвоенные цепочки ДНК.

Решение задачи в стикерных системах начинают с образования библиотеки запоминающих комплексов, т.е. с длинных одинарных цепочек, к каждой из которых присоединяют короткие стикеры, создавая частично сдвоенные цепочки. В этом заключается одно из отличий от предыдущей модели, где одноцепочечных промежутков быть не может. Далее используется ряд операций, аналогичных предыдущему методу, только здесь они используются уже не для процесса фильтрации, полученного на первом шаге решения, а для реализации нахождения самого решения.

Таким образом, вычисление в стикерной модели есть последовательность операций слить, разделить, включить и очистить. Входом и выходом вычисления, аналогично предыдущему пункту, являются пробирки. Чтение выхода состоит в выделении из итоговой пробирки одного запоминающего комплекса и определении присоединенных к нему стикеров, либо в сообщении, что в этой пробирке нет ни одного запоминающего комплекса. Рассмотрим понятия пробирки и операций над ними чуть подробней.

*Пробирка* – это мультимножество, элементами которого являются запоминающие комплексы.

Начальная пробирка здесь является библиотекой запоминающих комплексов. Более строго, (k, l) – библиотека, где $1 \le i \le k$, состоит из запоминающих комплексов с k подцепочками, последние k – l из которых выключены, тогда как первые l включены или выключены всеми возможными способами. Таким образом, (k, l) – библиотека, рассматриваемая как мультимножество, содержит $2^l$ различных типов запоминающих комплексов. В начальной пробирке первые l подцепочек запоминающих комплексов используются для входных данных, а остальные k – l подцепочек – для хранения промежуточных данных и результатов.

Следующие основные операции определены для данного типа пробирок.

*Слить*. Как и раньше две пробирки объединяются в одну. При этом запоминающие комплексы из двух пробирок на входе без изменений, присоединенных к ним стикеров, переходят в новое мультимножество, являющееся объединением двух данных.

*Разделить.* Эта операция создает по данной пробирке N и целому числу i, $1 \le i \le k$, две новых пробирки +(N, i) и –(N, i). Пробирка +(N, i) (соответственно –(N, i)) состоит из всех запоминающих комплексов исходной пробирки N, в которых включена (соответственно выключена) i-ая подцепочка.

*Включить.* Для данной пробирки N и целого i, $1 \le i \le k$, эта операция создает новую пробирку *включить*(N, i), в которой у каждого запоминающего комплекса из N включена i-ая подцепочка. Это означает, что подходящий стикер присоединяется к запоминающему комплексу, если i-ая подцепочка этого комплекса была выключена, но если i-ая подцепочка уже была включена, то комплекс не изменяется.

*Очистить.* Для данной пробирки N и целого i, $1 \le i \le k$, эта операция создает новую пробирку *очистить*(N, i), в которой у каждого запоминающего комплекса из N

выключена i-ая подцепочка. Это означает, что соответствующие стикеры удаляются из тех комплексов, где они имелись.

Вычислительная парадигма, ассоциированная со стикерной моделью, состоит в решении трудных задач путем полного перебора всех входов длины l, находящихся в начальной пробирке. Все возможные $2^l$ вариантов решений обрабатываются параллельно. В результате находится верное решение или не находится вообще, если решения не существует.

### C. Плиточная модель.

В лаборатории молекулярных вычислений в Калифорнийском технологическом институте под руководством Э. Винфри успешно разрабатываются методы синтеза различных поверхностей при помощи ДНК. В этих экспериментах переосмысливается само понятие вычисления. Оказывается, можно использовать двумерные плитки различной формы, которые могут взаимодействовать по локальным правилам (соединяться друг с другом), для того, чтобы получить в результате взаимодействия множества плиток желаемую глобальную структуру. При этом под вычислением понимается процесс создания такой структуры.

Общая идея модели заключается в принципе самосборки. В природе молекулярная самосборка дает начало огромному числу комплексов, включая кристаллы (такие как алмаз или бриллиант) и двойные спирали молекул ДНК. Т.е. рост таких структур контролируется на фундаментальном уровне естественными вычислительными процессами. Основная же проблема таких вычислений заключается в изучении принципов самоорганизации в естественных системах.

В работах Э. Винфри отработана методика перехода от двумерных плиток к молекулам ДНК. Существует задача об отыскании набора геометрических фигур на плоскости (плиток), которыми Евклидова плоскость может быть покрыта *только непериодическим образом*. В 1961 г. было показано, что невозможно создать алгоритм, который определяет, можно ли покрыть плоскость при помощи заданного набора плиток, или нет. Позже был предъявлен набор из 20426 плиток, которыми можно покрыть плоскость только непериодически. В дальнейшем количество плиток было сокращено сначала до 104, а затем и до 6, и, наконец, до двух.

Интересен следующий факт: Р. Пенроуз получил свой набор из 2-х плиток путем различных манипуляций *разрезания и склеивания* над набором Робинсона из 6 плиток.

В свете мысли об экспериментах Э. Винфри и мысли о задаче покрытия, в которых исходным материалом служат наборы плиток, которые затем преобразуются в молекулы ДНК, рождается идея о разработке парадигмы ДНК-вычислений именно в «плиточных терминах». При этом ДНК-вычислитель будет представлять собой клеточный автомат из клеток произвольной формы, а локальные правила взаимодействия клеток будут определяться их формой. С одной стороны, такой автомат будет дискретным, т.к. будет состоять из отдельных взаимодействующих плиток, и к нему будет применимо понятие шага. А с другой стороны, локальные правила задаются за счет непрерывной формы границы взаимодействующих плиток.

Рассмотрим эту модель чуть подробней. Как уже упоминалось, в рамках этой модели вычисления осуществляются благодаря самосборке квадратных плиток, каждая сторона которых промаркирована определенным образом. Различные маркеры представляют собой пути, по которым плитки могут быть связаны друг с другом. При этом сила (или липкость) связывания зависит от связывающей силы сопоставленной каждой из сторон. Правила, по которым система закодирована, основаны на выделении плиток со специфическими комбинациями маркеров и связывающих сил. Мы же в свою очередь

берем на себя обеспечение неограниченного числа каждой из плиток. Вычисления начинаются со специфической начальной плитки и продолжается последовательным добавлением одиночных плиток. Плитки связываются друг с другом, формируя растущий комплекс, представляющий собой состояние вычислений, только если их связывающие взаимодействия обладают достаточной силой (липкостью) (т.е. плитки склеиваются друг с другом таким образом, что полученный комплекс устойчив).

Данный подход к вычислениям сразу же обеспечивает возможность описания параллельных процессов, которые изначально присущи ДНК-вычислителю. При всей фантастичности данного подхода, нельзя не признать, что он несет значительный эвристический потенциал.

### D. Модель вычислений в терминах теории формальных языков.

Данная модель использует всю мощь теории формальных языков и представляет собой полную формализацию операций с ДНК. Разработанный теоретический базис вполне может быть использован для проектирования и разработки инструментальных средств: специального байт-кода, который имитирует работу ДНК-компьютера, интерпретатора этого байт-кода, языков высокого уровня, компиляторов в байт-код и т.д. То есть эта модель полностью приспособлена для обычного компьютера. Но при этом возникают и свои нюансы. В данной модели строго моделируется только свойство комплементарности молекул ДНК, свойство же массового параллелизма здесь никак не задействовано, так как, в действительности, его нельзя нигде использовать кроме как в реальном эксперименте.

В основе этой модели лежит так называемый язык перетасованных копий, который непосредственно связан со свойством комплементарности Уотсона-Крика, поскольку именно оно играет одну из центральных ролей в теории вычислений, основанных на ДНК.

Обозначим язык перетасованных копий в его основном варианте (над четырехбуквенным алфавитом {0, 1, $\bar{0}$, $\bar{1}$}) через TS.

Рассмотрим слово w над алфавитом {0, 1}, т.е. строку, состоящую из нулей и единиц. Пусть $\bar{w}$ - комплементарная строка, построенная из $\bar{0}$ и $\bar{1}$. Например, если w = 00101, то $\bar{w}$ = $\bar{0}\bar{0}\bar{1}\bar{0}\bar{1}$. Обозначим через sh(w, $\bar{w}$) множество всех слов, получаемых с помощью перетасовки слов w и $\bar{w}$, т.е. произвольной вставки букв слова $\bar{w}$ между буквами слова w без изменения порядка следования букв внутри этих слов. Например, каждое из слов $0\bar{0}0\bar{0}1\bar{1}0\bar{0}\bar{1}1$, $\bar{0}\bar{0}\bar{1}\bar{0}\bar{1}00101$, $00\bar{0}10\bar{0}\bar{1}\bar{0}\bar{1}1$ принадлежит множеству sh(w, $\bar{w}$), в то время как слово $0\bar{0}0\bar{0}1\bar{1}0\bar{1}\bar{0}1$ там не содержится. По определению, язык TS состоит из всех слов из sh(w, $\bar{w}$), где w пробегает множество всех слов над {0, 1}.

Следующий простой прием позволяет определить, принадлежит ли данное слово x, построенное из четырех букв 0, 1, $\bar{0}$, $\bar{1}$, языку TS. Сначала сотрем в x все буквы $\bar{0}$ и $\bar{1}$ и обозначим оставшееся слово через $x'$. Затем сотрем в x все буквы 0 и 1, а также все черточки над остающимися буквами и обозначим получившееся слово через $x''$. Тогда исходное слово x принадлежит языку TS в том и только в том случае, когда $x' = x''$.

Свяжем теперь «алфавит ДНК» и четырехбуквенный алфавит {0, 1, $\bar{0}$, $\bar{1}$} следующим образом:

А = 0, Г = 1, Т = $\bar{0}$, Ц = $\bar{1}$.

Если считать комплементарными буквы в парах (0, $\bar{0}$) и (1, $\bar{1}$), то при таком соответствии эта комплементарность есть в точности комплементарность Уотсона-Крика.

Существует несколько способов «считывать» слова из TS с нуклеотидов молекул ДНК.

*Первый способ:*

1. Рассматривается некоторая произвольная двойная цепочка ДНК.
2. Затем эта цепочка переписывается в терминах 0, 1, $\bar{0}$, $\bar{1}$, согласно указанному выше соответствию.
3. После этого из каждой одинарной цепочки (соответственно нижней и верхней) по очереди берутся буквы, образуя некоторую строку.
4. Полученная строка и будет строкой принадлежащей TS

*Второй способ:*

1. Рассматривается некоторая произвольная двойная цепочка ДНК.
2. Затем нуклеотиды А и Т отождествляются с 0 при появлении в верхней цепочке и с $\bar{0}$ при появлении в нижней цепочке; нуклеотиды Ц и Г отождествляются с 1 в верхней цепочке и с $\bar{1}$ в нижней. См. таблицу.
3. Далее, читая обе цепочки (соответственно нижнюю и верхнюю) данной двухцепочечной молекулы ДНК слева направо так, что скорость чтения каждой из цепочек недетерминирована и не зависит от скорости чтения другой цепочки, мы получаем строку из TS.

Одним из важных свойств языка TS является то, что он универсален. Согласно общепринятому тезису Черча-Тьюринга, любое вычисление может быть осуществлено машиной Тьюринга и, таким образом, вычисления характеризуются языками $L_0$, принимаемыми машинами Тьюринга. С другой стороны, каждый такой язык $L_0$ может быть представлен в виде $L_0 = f(TS)$, где f - так называемое ОПМ – отображение, зависящее от языка $L_0$. (Аббревиатура «ОПМ» происходит от слов «Обобщенная Последовательная Машина».) Таким образом, язык TS раз и навсегда фиксирован, в то время как отображение f меняется в зависимости от специфики языка $L_0$. Можно рассматривать отображение f как своего рода устройство ввода/вывода.

Для вычислений, основанных на ДНК, ситуация вполне аналогична. Комплементарность Уотсона-Крика раз и навсегда фиксирована и обеспечивает универсальность в том же смысле, что и TS. Основная же задача, ассоциированная с моделью вычислений в терминах теории формальных языков, состоит в нахождении наиболее приспособленных для ДНК-вычислений типов ОПМ – отображений f.

# Глава 2. Новая идеология решения NP-полных задач.

## • NP-полнота и ее задачи.

Здесь мы рассмотрим 3 интересных класса задач: P, NP и NPC (класс NP-полных задач).

Класс P состоит из задач, разрешимых в течение полиномиального времени работы. Точнее говоря – это задачи, которые можно решить за время $O(n^k)$, где k – некоторая константа, а n – размер входных данных задачи.

Класс NP состоит из задач, которые поддаются проверке в течение полиномиального времени. Имеется в виду, что если мы каким-то образом получаем «сертификат» решения, то в течение времени, полиномиальным образом зависящего от размера входных данных задачи, можно проверить корректность такого решения. Например, в задаче о гамильтоновом цикле с заданным ориентированным графом $G = (V, E)$ сертификат имел бы вид последовательности $(v_1, v_2, \ldots, v_p)$ из p вершин. В течение полиномиального времени легко проверить, что $(v_i, v_{i+1}) \in E$ при $i = 1, 2, \ldots, p-1$ и что $(v_p, v_1) \in E$.

Любая задача класса P принадлежит классу NP, поскольку принадлежность классу P означает, что ее решение можно получить в течение полиномиального времени, даже не располагая сертификатом.

Что же касается самих NP-полных задач, то до настоящего времени для них не разработано алгоритмов с полиномиальным временем работы, но и не доказано, что для какой-то из этих задач таких алгоритмов не существует. Этот так называемый вопрос P ≠ NP с момента своей постановки в 1971 году стал одним из самых трудных в теории вычислительных систем. Т.е. на данный момент не известно, выполняется ли равенство P = NP, но, по мнению большинства исследователей, классы P и NP – не одно и то же. Интуитивно понятно, что класс P состоит из задач, которые решаются быстро. Класс же NP состоит из задач, решение которых можно быстро проверить.

Особо интригующим аспектом NP – полных задач является то, что некоторые из них на первый взгляд аналогичны задачам, для решения которых существуют алгоритмы с полиномиальным временем работы. В качестве примера таких задач можно привести следующие:

1. Поиск самых коротких и самых длинных простых путей.
2. Эйлеров и Гамильтонов циклы.
3. 2- CNF и 3-CNF выполнимость.

Неформально задача принадлежит классу NPC (т.е. классу NP-полных задач), если она принадлежит классу NP и является такой же «сложной», как и любая задача из класса NP. Т.е. основное свойство этого класса заключается в том, что если хоть одну (любую) NP - полную задачу можно было бы решить в течение полиномиального времени, то и все задачи этого класса обладают полиномиально-временным решением, т.е. P = NP.

Большинство ученых, занимающихся теорией вычислительных систем, считают NP-полные задачи очень трудноразрешимыми, потому что при огромном разнообразии изучавшихся до настоящего времени NP-полных задач ни для одной из них пока так и не найдено решение в виде алгоритма с полиномиальным временем работы. Таким образом, было бы интересно найти способ решения этих задач за полиномиальное время.

Теперь рассмотрим основные и наиболее встречающиеся NP – полные задачи. Такие задачи возникают в различных областях: в булевой логике, в теории графов, в арифметике, при разработке сетей, в теории множеств и разбиений, при хранении и поиске информации, при планировании вычислительных процессов, в математическом программировании, в алгебре и теории чисел, при создании игр и головоломок, в теории автоматов и языков, при оптимизации программ, в биологии, в химии, физике и т.п.

✓ **Задача о клике.**

Клика неориентированного графа G = (V, E) – это подмножество $V' \subseteq V$ вершин, каждая пара в котором связана ребром из множества E. Другими словами, клика - это полный подграф графа G. Размер клики – это количество, содержащихся в нем вершин. Задача о клике – это задача оптимизации, в которой требуется найти клику максимального размера, содержащуюся в заданном графе. В соответствующей задаче принятия решения спрашивается, содержится ли в графе клика заданного размера k. Формальное определение клики имеет вид

Клика = {<G, k> : G – граф с кликой размера k}.

Простейший алгоритм определения того, содержит ли граф G = (V, E) с p вершинами клику размера k, заключается в том, чтобы перечислить все k-элементные подмножества множества V, и проверить, образует ли каждое из них клику. Время работы этого алгоритма равно $\Omega\ (k^2\binom{p}{k})$ и является полиномиальным, если k – константа. Однако в общем случае величина k может достигать значения, близкого к p/2, и в этом случае время

работы алгоритма превышает полиномиальное. Есть основания предполагать, что эффективного алгоритма для задачи о клике не существует.

✓**Задача о вершинном покрытии.**

Вершинное покрытие неориентированного графа G = (V, E) – это такое подмножество $V' \subseteq$ V, что если (u, v) $\in$ E, то u $\in V'$ либо v $\in V'$ (либо справедливы оба эти соотношения). Другими словами, каждая вершина «покрывает» инцидентные ребра, а вершинное покрытие графа G – это множество вершин, покрывающих все ребра из множества E. Размером вершинного покрытия называется количество содержащихся в нем вершин. Например, граф, изображенный на рис. 4. имеет вершинное покрытие {w, z} размера 2.

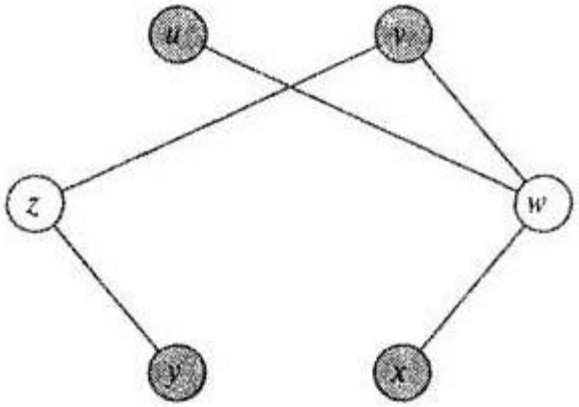


Рис. 4. Граф с вершинным покрытием размера 2.

Задача о вершинном покрытии заключается в том, чтобы найти в заданном графе вершинное покрытие минимального размера.

✓**Задача о гамильтоновых циклах.**

Задача поиска гамильтоновых циклов в неориентированном графе изучается уже более ста лет. Формально гамильтонов цикл неориентированного графа G = (V, E) – это простой цикл, содержащий все вершины множества V. Граф, содержащий все вершины множества V. Граф, содержащий гамильтонов цикл, называют гамильтоновым; в противном случае он является негамильтоновым. На самом деле не все графы являются гамильтоновыми. Например, двудольный граф с нечетным количеством вершин. Задача о гамильтоновых циклах заключается в том, что нужно определить, содержит ли граф G гамильтонов цикл.

✓**Задача о коммивояжере.**

В задаче о коммивояжере, которая тесно связана с задачей о гамильтоновом цикле, коммивояжер должен посетить n городов. Моделируя задачу в виде полного графа с n вершинами, можно сказать, что коммивояжеру нужно совершить тур, или гамильтонов цикл, посетив каждый город по одному разу и завершить путешествие в том же городе, из которого он выехал. С каждым переездом из города i в город j связана некоторая стоимость c(i, j), выражающаяся целым числом, и коммивояжеру нужно совершить тур таким образом, чтобы общая стоимость (т.е. сумма стоимостей всех переездов) была минимальной.

✓**Задача о сумме подмножества.**

Данная задача принадлежит к разряду арифметических. В задаче о сумме подмножества задается конечное множество S $\subset$ N и целевое значение t $\in$ N. Спрашивается, существует ли подмножество $S' \subseteq$ S, сумма элементов которого равна t. Например, если

S = {1, 2, 7, 14, 49, 98, 343, 686, 2409, 2793, 16808, 17206, 117705, 117993}

а t = 138457, то решением является подмножество

$S'$ = {1, 2, 7, 98, 343, 686, 2409, 17206, 117705}.

✓**Задача об изоморфизме подграфу.**

Задаются два графа ($G_1$ и $G_2$) и спрашивается, изоморфен ли граф $G_1$ какому-нибудь подграфу графа $G_2$.

✓**Задача 0-1 целочисленного программирования.**

Задается целочисленная матрица A размером m $\times$ n и целочисленный m-компонентный вектор b и спрашивается, существует ли целочисленный n-компонентный вектор x, элементы которого являются элементами множества {0, 1}, удовлетворяющий неравенству Ax $\leq$ b.

✓**Задача целочисленного линейного программирования.**

Эта задача похожа на задачу 0-1 целочисленного программирования, описанную выше, но в ней компоненты вектора x могут быть любыми целыми числами, а не только нулем и единицей.

✓**Задача о разделении множества.**

Здесь в качестве входных данных выступает множество чисел S. Спрашивается, можно ли это множество чисел разбить на два подмножества A и $\bar{A}$ = S – A таким образом, чтобы $\sum_{x \in A} x = \sum_{x \in \bar{A}} x$.

✓**Задача о самом длинном простом цикле.**

Пусть задан некоторый граф. Тогда задача заключается в том, чтобы найти в этом графе простой цикл (т.е. без повторения вершин) максимальной длины.

✓**Половинная задача о 3-CNF выполнимости.**

Задается 3-CNF формула $\tau$ с n переменными и m подвыражениями в скобках, где m – четное. Нужно определить, существует ли набор значений переменных формулы $\tau$, при котором результат ровно половины выражений в скобках равен 0, а результат второй половины этих выражений равен 1.

✓**Задача о минимальном покрытии множества.**

Дано ограниченное множество S = {2, …, p}, состоящее из (p – 1) элементов и ограниченная коллекция {$C_1$, …, $C_q$} подмножеств множества S. Требуется найти наименьшее подмножество I из {1, 2, …, q} таких что $\bigcup_{i \in I} C_i$ = S.

✓**Задача о раскраске графов.**

Если задан неориентированный граф G = (V, E), то его k-раскрашиванием называется функция c : V $\rightarrow$ {1, 2, …, k}, такая что c(u) $\neq$ c(v) для каждого ребра (u, v) $\in$ E. Другим словами, числа 1, 2, …, k представляют k цветов, и смежные вершины должны быть разного цвета. Задача о раскрашивании графов заключается в том, чтобы определить минимальное количество цветов, необходимых для раскрашивания заданного графа.

Выше мы привели основные и самые интересные NP-полные задачи. В следующем пункте мы попытаемся решить некоторые из этих задач в рамках ДНК вычислений, чтобы показать эффективность этого метода решений трудоемких задач на реальных примерах.

## • Оценки сложности некоторых NP-полных задач.

Перед тем как начать разбирать примеры решения NP-полных задач в рамках моделей, описанных в предыдущем пункте, хотелось бы привести оценки сложности некоторых из них.

Таблица 1 описывает результаты моделирования различных классических вычислительных моделей в рамках ДНК-вычислений. Каждый результат имеет две оценки:

1. Сколько это решение занимает биологических шагов?
2. Сколько ДНК нитей оно использует?

Оценочные алгоритмы по этим атрибутам таблицы не новые. Они уже появлялись в области параллельных алгоритмов. Параллельный алгоритм должен быть быстрым, т.е. занимать несколько параллельных шагов. Однако, он также должен использовать относительно мало параллельных процессоров. Таким образом, алгоритм, который занимает O(log n) шагов, но использует $n^4$ процессоров – не практичен. С тех пор, как молекулы ДНК стали использоваться, как параллельные процессоры, было бы естественно оценивать данные задачи в подобной двойной манере (т.е. с помощью двух атрибутов).

Под числом ДНК нитей мы понимаем число одинарных цепочек, которые могут появляться в одной и той же тестовой пробирке в течение периода выполнения алгоритма. Это количество приблизительно соответствует объему тестовой пробирки. Чтобы быть более точными, нам также следовало бы включить сюда и длину этих нитей; однако это не очень важно, так как длина данных нитей обычно линейна размеру проблемы, пока число нитей экспоненциально.

Под числом шагов мы просто обозначаем полное число биологических операций в течение периода выполнения алгоритма. Мы здесь не делаем различий между разными операциями, даже не смотря на то, что время нужное для каждой из них может отличаться.

| Номер | Проблема | Биологические шаги | Нити |
|---|---|---|---|
| **1.** | Направленный гамильтонов путь | O(n) | n! |
| **2.** | КНФ - выполнимость | O(s) | $2^n$ |
| **3.** | Максимальная клика | O(s) | $2^n$ |
| **4.** | 1-ленточная недетерминированная машина Тьюринга | O(t) | $2^N$ |
| **5.** | Клеточный автомат | 1 | $t \cdot S$ |
| **6.** | 3 - раскраска графа | O(n) | $2^n$ |
| **7.** | Изоморфизм подграфа | $O(\|V_s\|)$ | $2^s$ |

Таблица 1. Оценки сложности некоторых NP-полных задач.

Приведем разъяснения обозначений из указанной таблицы:

**1.** Граф состоит из n вершин и для решения проблемы требуется n! нитей.

**2.** Здесь s – размер пропозициональной формулы, а n – количество переменных.

3. В данном случае через s обозначается размер клики, а через n число входных переменных.
4. Здесь t обозначает время, а N – число недетерминированных бит, используемых машиной Тьюринга.
5. Данная конструкция была предложена Винфри и показывает, как сложные ДНК структуры могут быть использованы для моделирования клеточного автомата. При этом число нуклеотидов, используемых данной ДНК структурой пропорционально произведению интервала, используемого автоматом (S) и числа генераций решетки этого автомата с новыми состояниями (t).
6. Здесь n – количество вершин в графе.
7. В данном случае $|V_s|$ - количество вершин в основном графе $G_1$.

## • Решение NP-полных задач средствами ДНК.

### A. Модель параллельной фильтрации.

✓ **Решение задачи о 3 – раскраске графа.**

*Проблема:* Три цвета.

Дан граф G = (V, E), найти трех цветное покрытие, если оно существует, в противном случае вернуть пустое значение.

*Решение:*

Входные данные: Пусть задано множество U, состоящее из всех строчек вида $p_1c_1p_2c_2\ldots p_nc_n$ где n = |V| число вершин в графе. Здесь для всех i, $p_i$ однозначно кодирует «позицию i», а каждое $c_i$ представляет собой какой-либо из трех «цветов» 1, 2 или 3. Каждая такая строка представляет собой одно из возможных присваиваний цветов вершинам графа так, что для каждого i цвет $c_i$ сопоставлен вершине i.

Алгоритм:

**for** j = 1 до n **do**
  **begin**
    *Размножить* (U, {$U_1$, $U_2$, $U_3$})
    **for** i = 1, 2 и 3, и для всех k таких что (j, k) ∈ E
      **in parallel do** *Извлечь* ($U_i$, {$p_j \neq i$, $p_k i$})
    *Слить* ({$U_1$, $U_2$, $U_3$}, U)
  **end**
*Обнаружить* (U)

Сложность: O(n) (параллельное время).

Число ДНК нитей: $2^n$

После j-ой итерации в цикле **for** вычисление гарантирует, что в оставшихся строках вершина j (при этом она может быть окрашена в один из трех цветов 1, 2 или 3 в зависимости от множества $U_i$ в котором она сохранилась в данный момент) не будет иметь смежных вершин, схожих с ней по цвету. Таким образом, когда алгоритм

закончится, множество U будет окрашено только разрешенными цветами, если такие существуют. В действительности каждый разрешенный цвет будет представлен в множестве U.

*Блок схема:*

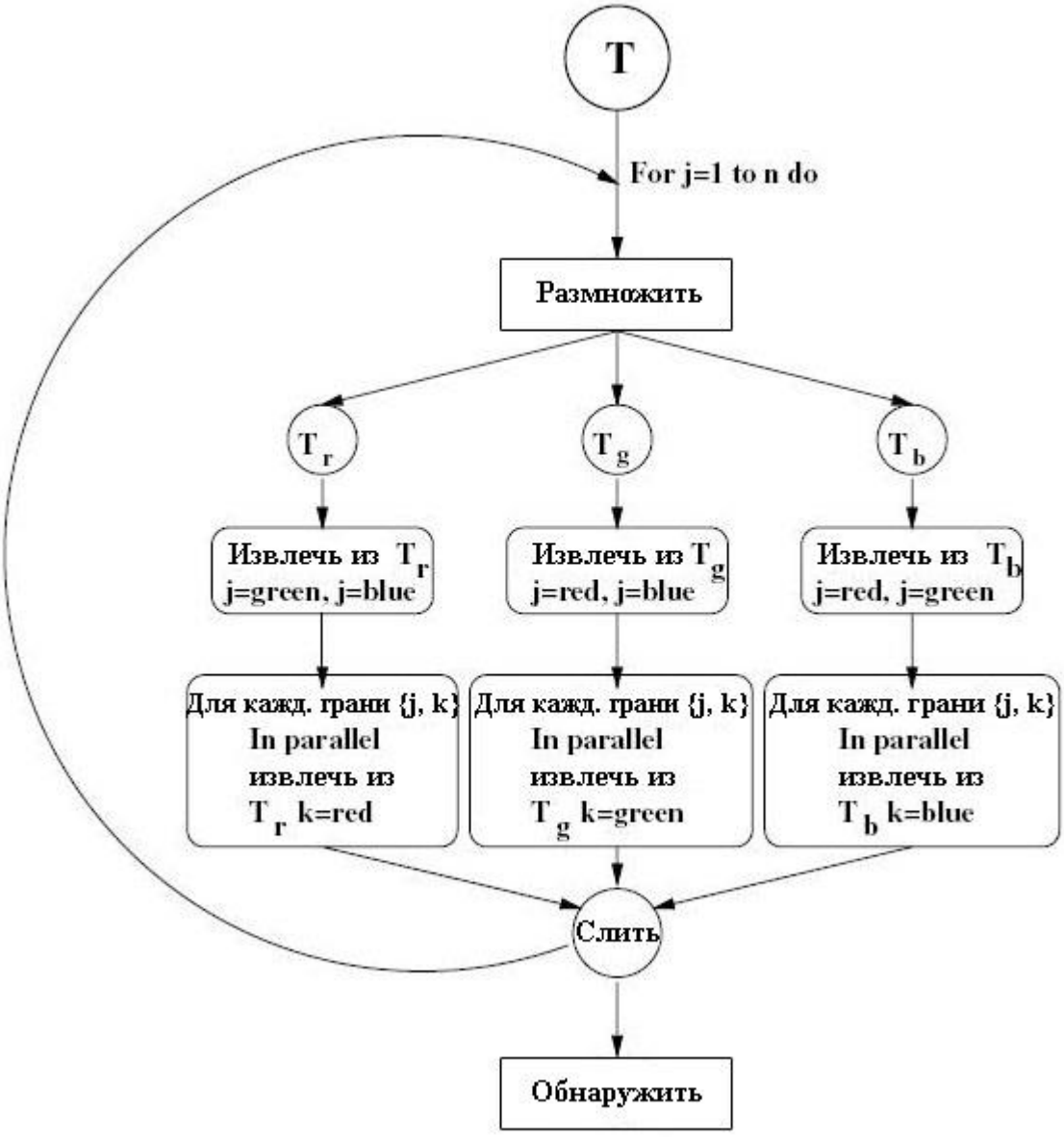


Рис. 5. Блок схема алгоритма для 3-раскраски графа.

- ✓ **Решение задачи об изоморфизме подграфу.**

*Проблема:* Изоморфизм подграфу.

Является граф $G_2 = (V_2, E_2)$ подграфом графа $G_1 = (V_1, E_1)$? Через $\{v_1, v_2, \ldots, v_s\}$ мы обозначим вершины множества $G_1$; аналогично через $\{u_1, u_2, \ldots, u_t\}$ обозначим вершины множества $G_2$, где, не умоляя общности, будем считать, что $t \leq s$.

*Решение:*

Входные данные: Пусть заданное множество U представляет собой множество $P_s$ всех перестановок целых чисел от 1 до s. Целое число i в позиции $p_k$ в такой перестановке интерпретируется как строка, представляющая собой возможное решение проблемы в рассматриваемой вершине i на шаге k. Для $1 \leq j \leq t$ строка $p_1i_1p_2i_2 \ldots p_si_s$ из множества $P_s$ интерпретируется как связь вершины $p_j \in \{u_1, u_2, \ldots, u_t\}$ с вершиной $i_j \in \{v_1, v_2, \ldots, v_s\}$. Алгоритм написан таким образом, чтобы извлекать какую-либо из строк, которая устанавливает соответствие между вершинами из $V_1$ и $V_2$ таким способом, который не отражает требования, что если $(p_s, p_t) \in E_1$ тогда $(i_s, i_t) \in E_2$.

Алгоритм:

**for** j = 1 до t – 1 **do**
  **begin**
    *Размножить* (U, {$U_1, U_2, \ldots, U_t$})
    **for** всех l, $j < l \leq t$ таких что $(p_j, p_l) \in E_2$ и $(ij, i_l) \notin E_1$
      **in parallel do** *Извлечь* ($U_j$, {$p_ii_l$})
    *Слить* ({$U_1, U_2, \ldots, U_t$}, U)
  **end**
*Обнаружить* (U)

Сложность: $O(|V_s|)$ (параллельное время)

Число ДНК нитей: $2^s$

Для каких-либо оставшихся строчек первые t пар $p_it_i$ представляют собой взаимно-однозначную связь вершин из $G_1$ с вершинами из $G_2$, указывающие на подграф графа $G_1$, изоморфный графу $G_2$. Если операция *Обнаружить* (U) вернет пустое значение, тогда $G_2$ не является подграфом графа $G_1$.

✓ **Решение задачи о максимальной клике.**

*Проблема:* Максимальная клика

Дан граф G = (V, E). Требуется определить наибольшее i, такое, что $K_i$ подграф графа G. Здесь $K_i$ полный подграф с i вершинами – максимальная клика.

*Решение:*

Алгоритм: Решение находится в процедуре параллельного вычисления из алгоритма предыдущей задачи для пар графов (G, $K_i$) для $2 \leq i \leq n$. Наибольшее значение i с непустым результатом (т.е. с непустым множеством) и будет решением задачи.

Сложность: O(|V|) (параллельное время)

Число ДНК нитей: $2^n$

**B. Стикерная модель.**

✓ **Решение задачи о минимальном покрытии множества.**

*Проблема:* Минимальное покрытие.

Опишем эту задачу в неформальной форме. Пусть задано множество разных предметов и множество сумок, содержащих различные подмножества этого множества предметов. Тогда какое наименьшее число сумок должен держать человек, чтобы гарантировать, что он или она держат, по крайней мере, по одному из заданных предметов.

Пусть множество предметов будет S = {2, …, 5}, а коллекция сумок будет {$C_1$,…, $C_4$}. Следовательно, требуется найти минимальное подмножество I из {1,…, 4}, таких что $\bigcup_{i \in I} C_i$ = S.

*Иллюстрация:*

Задача проиллюстрирована на рис. 6., где (a) – множество из 5 пронумерованных предметов, а (b) – 4 разных сумки, каждая из которых содержит различные подмножества предметов заданного множества.

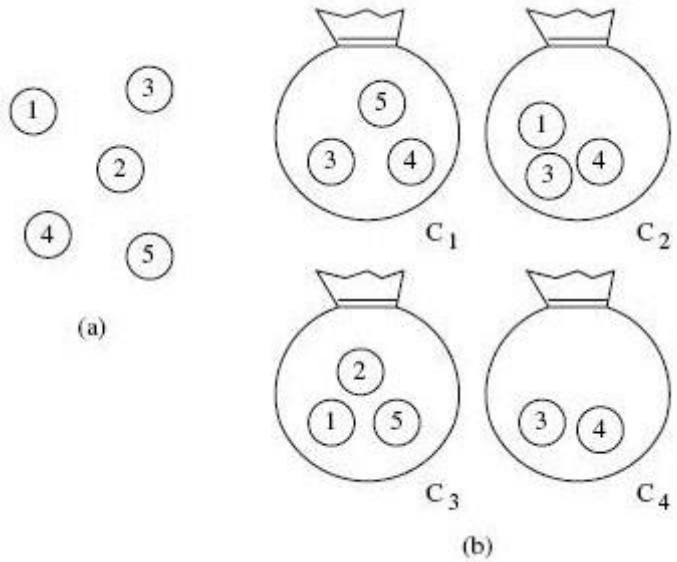


Рис. 6. (a) пять возможных предметов, (b) 4 разных сумки.

*Решение:*

Запоминающие цепочки имеют k = p + q подцепочек, а начальная пробирка $N_0$ содержит (p + q, q) библиотеку. Каждая запоминающая цепочка представляет собой подмножество сумок, взятое из множества C с первыми q подцепочками кодирующими наличие или отсутствие сумок. Если подцепочка i присоединена, тогда сумка $C_i$ присутствует в подмножестве, если подцепочка i выключена, тогда $C_i$ отсутствует. Например, запоминающая цепочка с включенными 1, 3 и 4 подцепочками 1011 кодирует подмножество, содержащее 1, 3 и 4 сумки, т.е. $C_1$, $C_3$, и $C_4$. Если же включена только одна подцепочка, например 0010, тогда она кодирует подмножество, содержащее только одну сумку $C_3$. Цепочки наклейки (или рабочие подцепочки) представляют собой

подмножество предметов, содержащееся в сумках. Например, в соответствии с рис. 6., запоминающая цепочка, кодирующая выражение 0101**10110** интерпретируется следующим образом: первые четыре элемента (0101) обозначают, что в подмножестве содержатся 2 и 4 сумки, а объединение $C_2 \cup C_4 = \{1, 3, 4\}$ указывает на те предметы, которые содержатся в этих сумах (см. **10110**).

Конечно, когда создана начальная библиотека запоминающих комплексов все рабочие подцепочки выключены, поэтому вначале мы должны сопоставить им подходящие значения. Это достигается следующим образом: для заданной запоминающей цепочки берем полное подмножество предметов, закодированное всеми сумками, в которых оно содержится и включаем подцепочки, представляющие каждый предмет из этого подмножества. Мы представим содержимое начальной пробирки $N_0$ после сборки цепочек в таблице 2. Заметьте, что включение обозначается 1, а выключение 0.

Допустим мы имеем это множество цепочек и мы желаем сохранить только те из них, которые кодируют подмножество сумок, которое покрывает множество S (т.е. те подмножества, которые содержат по одному из каждого объекта в исходном множестве). Мы достигаем этого, отбраковывая те цепочки, в которые не включены по каждой из 5 рабочих подцепочек (представляющие предметы). Результат этих операций представлен в таблице 3.

| Шаг | $C_1$ | $C_2$ | $C_3$ | $C_4$ | 1 | 2 | 3 | 4 | 5 | Подмножество |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Пустое |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | (3, 4, 5) |
| 3 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | (1, 2, 5) |
| 4 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |
| 5 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | (1, 3, 4) |
| 6 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | (1, 3, 4) |
| 7 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |
| 8 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |
| 9 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | (3, 4, 5) |
| 10 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | (3, 4, 5) |
| 11 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |
| 12 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |
| 13 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | (1, 3, 4, 5) |
| 14 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | (1, 3, 4, 5) |
| 15 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |
| 16 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |

Таблица 2. Начальная пробирка после сборки цепочек.

| Шаг | $C_1$ | $C_2$ | $C_3$ | $C_4$ | 1 | 2 | 3 | 4 | 5 | Подмножество |
|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |
| 7 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |
| 8 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |
| 11 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |
| 12 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |
| 15 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |
| 16 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | (1, 2, 3, 4, 5) |

Таблица 3. Начальная пробирка после отбраковки.

Теперь, создав, указанным выше способом, множество покрытий, следующей задачей будет найти среди них то покрытие, которое использует наименьшее число сумок, если такое существует. Это достигается сортировкой начальной пробирки $N_0$ в пробирки с номерами $N_0$, $N_1$, …, $N_q$, где пробирка $N_i$ содержит запоминающие цепочки, кодирующие покрытия, использующие i сумок. Этот процесс сортировки может быть представлен следующим образом: вообразите строку пробирок, вытянутых слева направо начиная с пробирки $N_0$ и заканчивая пробиркой $N_q$. Затем мы осуществляем цикл, поддерживая счетчик i со значениями от 1 до 4, каждый раз сдвигая вправо одну пробирку с цепочками, кодирующими сумку i. Этот процесс показан на рис. 7. Заметьте, например, как цепочка (3, 4) – самая левая в пробирке $N_0$ смещается вправо в пробирку $N_1$, когда счетчик достигает значения 3. Продолжая сортировку таким образом, мы заканчиваем вычисление, получив множество пробирок, где пробирка $N_i$, $i \geq 1$ содержит только те цепочки, которые кодируют покрытия, использующие i сумок. Этот процесс может быть осуществлен с помощью электронной версии гель электрофореза, где сильные цепочки (те, которые кодируют большее число сумок) сдвигаются дальше вправо, чем слабые цепочки.

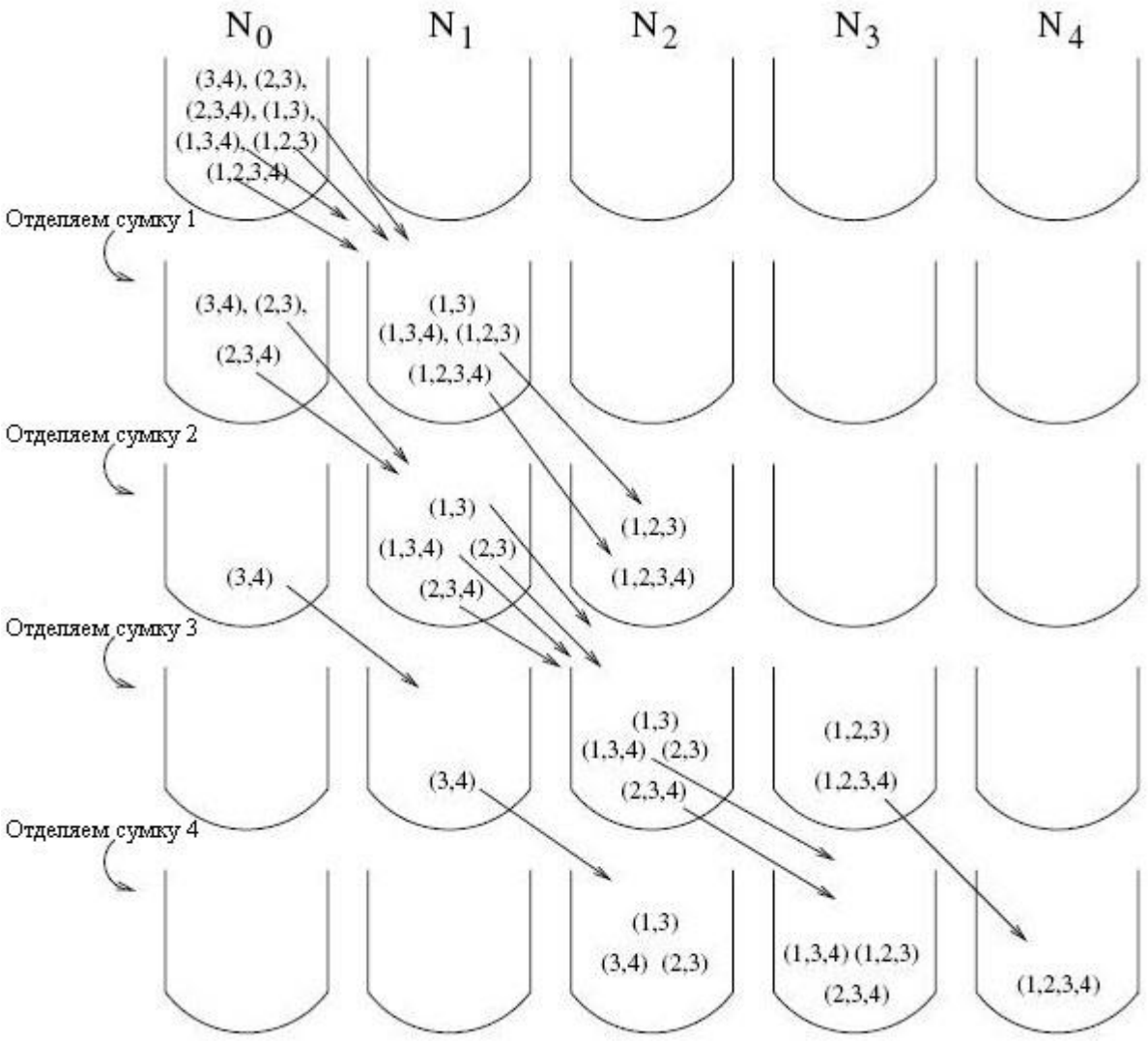


Рис. 7. Процедура сортировки.

Затем мы начинаем с пробирки с самым маленьким номером (т.е. $N_0$) и читаем ее содержание (если таковое имеется). Если эта пробирка пустая, то мы передвигаемся к следующей за ней по номеру пробирке и т.д. пока не найдем пробирку, которая содержит покрытие. В данном случае (см. рис. 7) пробирка $N_2$ содержит три покрытия, каждое из которых содержит по 2 сумки. Формально этот алгоритм в рамках стикерной модели можно представить следующим образом.

Алгоритм:

Инициализация (p, q) библиотеки в пробирке $N_0$
**for** i = 1 до q **do begin**
  $N_0$ ← *Разделить* (+($N_0$, i), -($N_0$, i))
    **for** j = 1 до |$C_i$|
      *Включить* (+($N_0$, i), $q + c_i^j$)
    **end for**
  $N_0$ ← *Слить* (+($N_0$, i), -($N_0$, i))
**end for**

Эта секция представляет собой объект, идентифицирующий подцепочки. Заметим, что выражение $c_i^j$ обозначает j-ый элемент множества $C_i$.

Сейчас мы выделяем из большинства используемых только те запоминающие комплексы, где включена каждая из последних p подцепочек.

**for** i = q + 1 до q + p **do begin**
  $N_0$ ← +($N_0$, i)
**end for**

Сейчас мы сортируем оставшиеся цепочки в соответствии с тем, как много сумок они кодируют.

**for** i = 0 до q – 1 **do begin**
  **for** j – 1 **down to** 0 **do begin**
    *Разделить* (+($N_j$, i+1), -($N_j$, i+1))
    $N_{j+1}$ ← *Слить* (+($N_j$, i+1), $N_{j+1}$)
    $N_j$ ← -($N_j$, i+1)
  **end for**
**end for**

Строка 3 здесь разделяет каждую пробирку в соответствии со значением i, а строка 4 осуществляет правый сдвиг выделенных цепочек. Затем мы ищем финальное решение.

Read $N_1$
**else if** empty **then** read $N_2$
**else if** empty **then** read $N_3$
**else if** empty **then** read $N_4$

## C. Плиточная модель

### ✓Задача реализации счетчика

*Проблема:* Требуется реализовать простейший алгоритм – счетчик.

*Решение:*

Пусть задано множество из 7 различных плиток в рамках системы, изображенной на рис. 8.: здесь мы имеем 4 управляющих плитки (промаркированы 1 или 0) $r_0$, $r_1$, $r_2$, $r_3$, две граничных плитки (промаркированы «L» и «R») и начальная плитка («S») (рис. 8.а.). Стороны этих плиток, изображенные одиночной линией, имеют связывающую силу 1, а стороны, которые изображены 2-ой линией, соответственно связывающую силу 2. Утолщенные линии имеют связывающую силу 0.

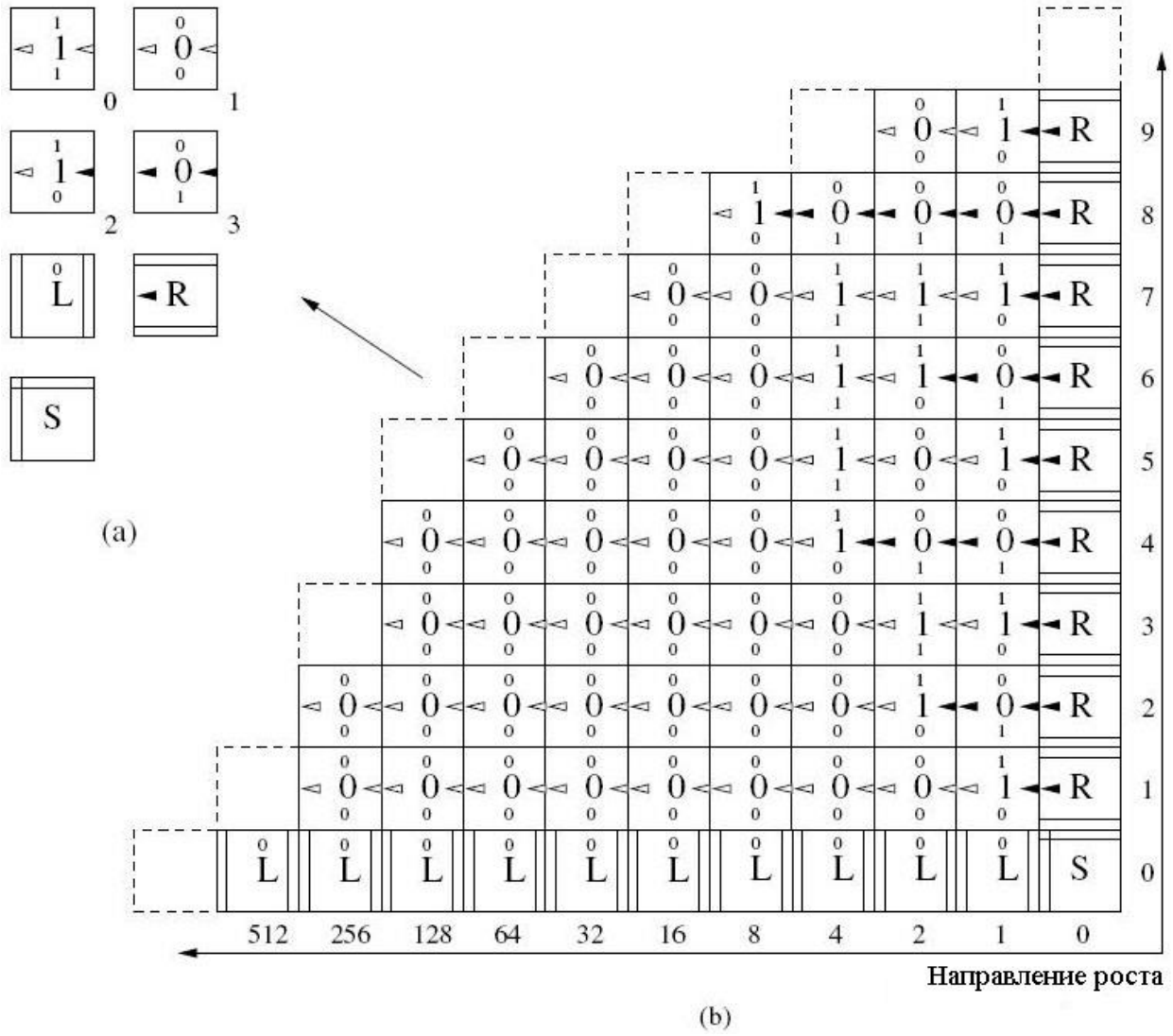


Рис. 8. (a) Заданные плитки, (b) Прогрессия роста сложности.

Мы фиксируем следующее важное ограничение: плитка может быть добавлена к сборке только, если она устанавливается в место, обладающее связывающей силой 2. Вдобавок, плитка с промаркированными сторонами (т.е. управляющая плитка) может

быть добавлена только, если маркеры ее стороны сопоставимы с маркерами, предлагаемого ей соседа. Понятно, что две управляющие цепочки в изоляции не могут связаться друг с другом, так как сила связывания между ними может быть только 1. Строясь с начальной плитки рабочая платформа, состоящая из L и R плиток, образуется, чтобы обеспечить сборку, т.е. структуру, появляющуюся благодаря связывающим силам, ассоциированным с их сторонами. Вообразите ситуацию в начале этого процесса, когда сборка состоит из одной начальной плитки S, одной плитки L и одной R. Единственная плитка, которая промаркирована сопоставимо со сторонами соседей (т.е. L и R плиток) это управляющая плитка $r_2$, поэтому именно она и добавляется к комплексу.

Сборка постепенно растет справа налево, от строчки к строчке, с плитками в строчке, представляющими удвоенные целые числа. Рост сборки изображен на рис. 8.b. с пространством, которое может быть заполнено плитками на следующей итерации, изображенной прерывистыми линиями. Заметьте, что рост комплекса ограничен только количеством плиток и границами, для которых северная и западная стороны сборки остаются в поле зрения во время ее роста. Заметьте также, что некоторая строка n не может увеличиться влево, пока n – 1 строка не увеличится, по крайней мере, на это же расстояние.

Таким образом, здесь показано, что для связывающей силы 2, может быть смоделирован одномерный клеточный автомат и поэтому эта самосборка универсальна.

### D. Модель в терминах теории формальных языков.

Для данной модели мы не будем приводить примеров ее использования, так как эта модель все-таки больше разработана для моделирования ДНК вычислений на обычном компьютере, а он в свою очередь не подразумевает решения трудоемких задач в общем виде. А основная заслуга реальных ДНК вычислений с использованием ДНК компьютера как раз таки и заключается в возможности решения трудоемких задач в общем виде.

# Часть 2. Обучающая система DNA-Learning

## ✓ Общее описание обучающей системы

Система DNA-Learning предназначена для обучения пользователей различным дисциплинам. В данном случае мы приводим пример обучения такой теме, как ДНК-вычисления и решение различных трудоемких задач с использованием этих молекул. Рассматриваемая система разрабатывалась с целью создания как можно более легкой в плане управления и как можно более понятной в плане интерфейса программы для широкого круга пользователей, начиная с людей, плохо разбирающихся в компьютерах и заканчивая продвинутыми пользователями.

Общая суть комплекса заключается в следующем. Пользователь регистрируется в программе, после чего ему становятся доступны все ее функции. В нашем случае в программе доступны 4 раздела: «История ДНК-вычислений», «Общие сведения о молекулярных вычислениях», «Модель параллельной фильтрации» и «Стикерная модель вычислений». На первом шаге доступ к 2, 3 и 4 разделам закрыт. Для того, чтобы перейти к следующему разделу пользователю надо прочитать и изучить материал предыдущего раздела, а также выполнить один из тестов, предлагаемых в этом разделе. Если пользователь все сделал правильно и получил проходной балл, то ему становится доступен следующий раздел. Таким образом, переходя от одного раздела к другому происходит последовательное изложение учебного материала и примеров по его

использованию. Когда обучаемый пройдет весь материал, он может распечатать прогресс его обучения, где представлена вся информация о том, что он изучил и какие оценки получил. Если обучаемому не понравятся свои достижения, то обучение можно начать заново.

Программа имеет следующий простой интерфейс.

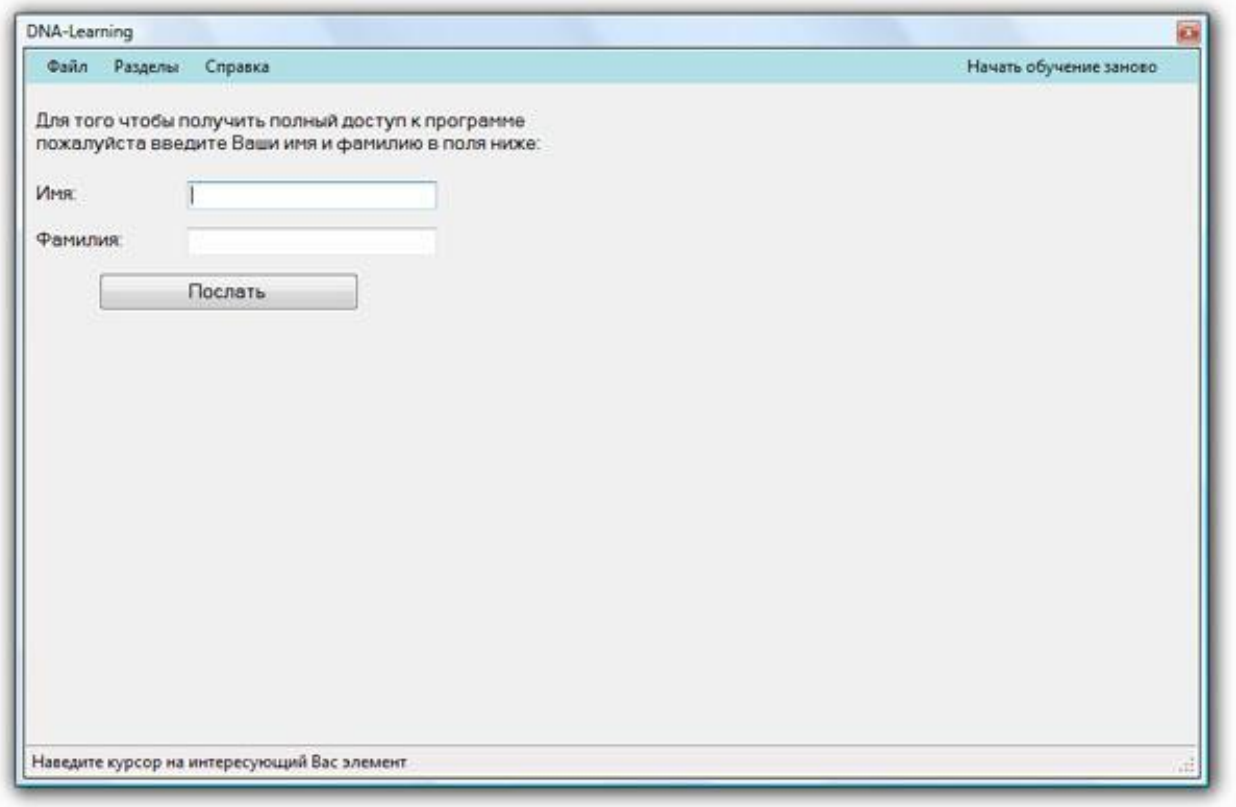


Рис. 9. Окно начальной загрузки программы.

Нетрудно вдеть, что программа разрабатывалась для систем Windows Vista и Windows 7. Но также проводилось ее тестирование и под систему Windows XP. Никаких неполадок выявлено не было. Таким образом, эту программу можно использовать на всех самых популярных на данный момент операционных системах.

Теперь разберем весь функционал программы, доступный на данный момент ее развития.

Главное меню программы имеет следующие категории:

1. Файл
   - Открыть
   - Сохранить
   - Сохранить как
   - Печать прогресса
   - Выход
2. Разделы
   - История
   - Общие сведения
   - Модель параллельной фильтрации
   - Стикерная модель
3. Справка

- Прогресс обучения
- Просмотр справки
- О программе

4. Начать обучение заново

Категория «Файл» содержит основные функции управления программой. Функция «Открыть» позволяет пользователю перейти к сохраненному им ранее документу, который сохраняется соответственно при нажатии на кнопку «Сохранить». Под сохраненным документом в первую очередь понимается сохранение имени и фамилии зарегистрированного пользователя а также его достижений в обучении. Пример файла сохранения будет приведен чуть позже. Если документ открыт в первый раз и еще не сохранялся, то функция «Сохранить» предложит обучаемому ввести имя для сохраняемого документа, в противном случае она просто пересохранит предыдущие данные на текущие. Функция «Сохранить как» всегда предлагает пользователю выбрать имя для его сохранения. Это имя может быть таким же, как и прошлое, либо новым, если обучаемый хочет сохранить текущий прогресс в новом файле. Нажатие на кнопку «Печать прогресса» пошлет на печать текущие достижения пользователя. Также как и с видом файла сохранения, пример печати прогресса будет приведен в следующем пункте. Нажатие на кнопку «Выход» просто закрывает программу.

В категории «Разделы», как уже упоминалось, пользователю доступны пройденные им разделы, а также новый раздел. При попытке войти в еще не доступный ему раздел появляется окно исключения, сообщающее обучаемому о невозможности выполнения этого действия.

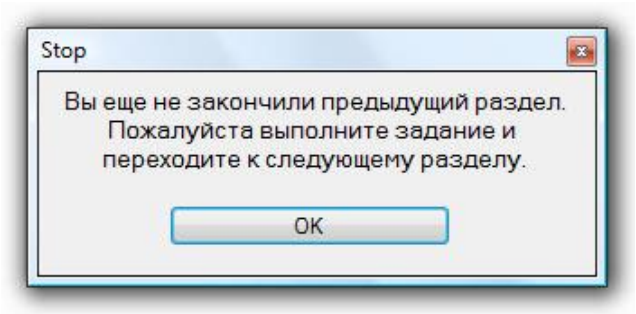


Рис. 10. Окно исключения.

Категория справка содержит справочную информацию по использованию программы. А именно, при нажатии на кнопку «Прогресс обучения» пользователю выводится информация о его достижениях, пройденных разделах и количестве баллов, полученных за выполненные к ним тесты. «Просмотр справки» знакомит обучаемого с основными возможностями программы а также дает ответы на вопросы о том, что и как делать. Функция «О программе» выводит основную информацию о комплексе DNA-Learning.

При нажатии на кнопку «Начать обучение заново» происходит обнуление достижений пользователя и количества открытых разделов. После этого обучение начинается заново.

Сама программа разбита на три основных части. Первая из них выводит текстовую информацию, т.е. текст справки, прогресса, соответствующих разделов и т.д. Этот блок располагается в верхней части программы. Следующий блок является графическим, т.к. выводит все доступные к открытому разделу фотографии, а также их список для быстрого перемещения от одной из них к другой. Стоит заметить, что изначально фотографии выводятся не в полном размере, а в уменьшенном для краткого просмотра. Для того чтобы увеличить фотографию достаточно просто на нее нажать и тогда уже будет показан ее

полноразмерный вариант. Для возвращения к исходному уменьшенному размеру надо кликнуть по фотографии еще раз. Этот блок располагается в нижней левой части программы. Следующим и последним блоком является блок заданий. Он располагается в нижней правой части программы и представляет собой ряд кнопок для перехода к соответствующему открытому разделу теста. Каждый из рассмотренных блоков доступен только в случае его необходимости. Т.е. если, допустим, в разделе нет фотографий, то блок фотографий просто отключается. Или мы вошли в справочный раздел, к которому не прилагается никаких заданий, то и блок заданий также будет отключен, выдав сообщение, что доступных заданий не имеется.

Следующей и последней частью программы является статусная строка, располагающаяся в самом низу программы. Она предназначена для вывода подсказок пользователю. Т.е. при наведении на какой-либо элемент программы там отображается справочная информация по этому элементу для лучшего ориентирования обучаемого.

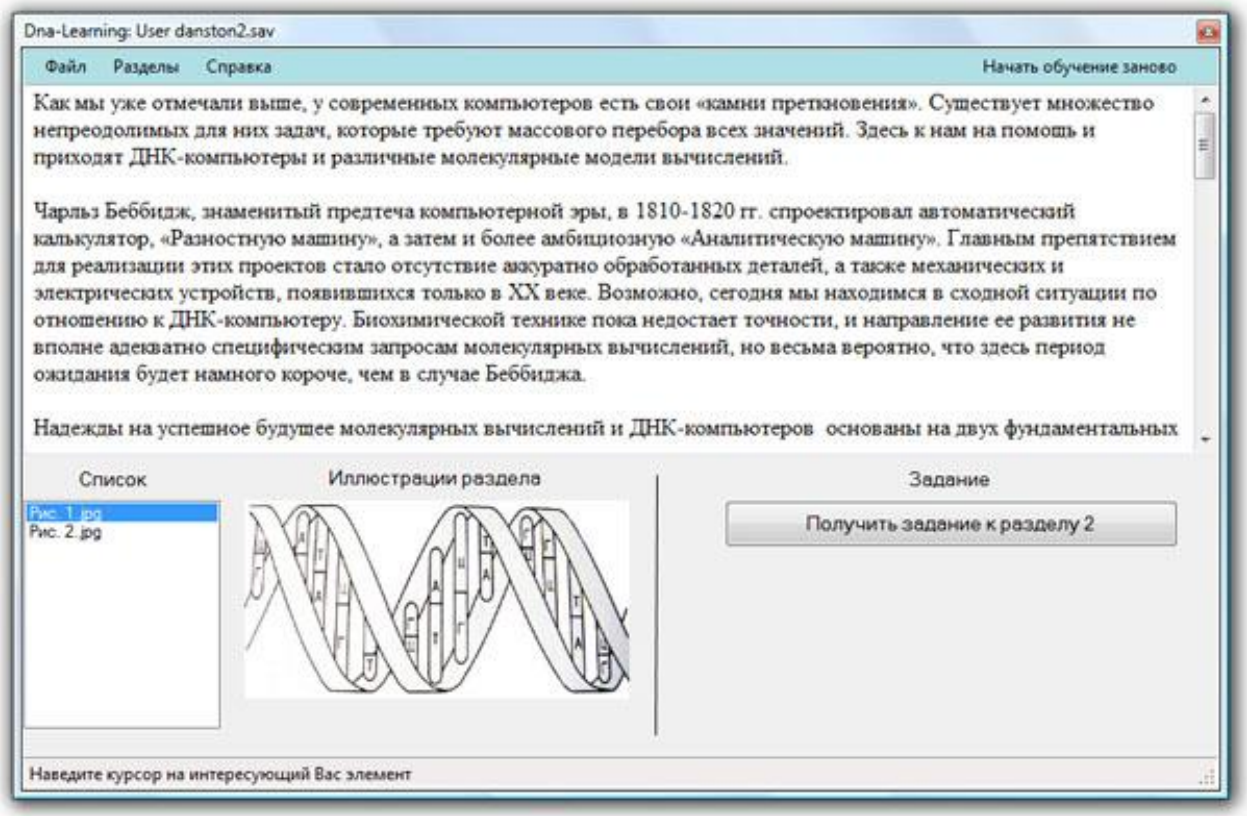


Рис. 11. Общий вид программы с открытым разделом.

## ✓ Структура файлов

Теперь разберемся со структурой файлов и заполнением системы информацией. Данная система использует три типа файлов: соответственно с расширениями .sav, .txt, .xml. Первый из них служит для сохранения информации и прогресса пользователя, второй для подготовки текстовой информации по разделам и справочных материалов, а третий для создания тестов к разделам программы. Разберем их более подробно.

Структура файла с расширением .sav имеет следующий вид

true!false!false!false\ДА!НЕТ!0\НЕТ!НЕТ!0\НЕТ!НЕТ!0\НЕТ!НЕТ!0\Имя!Фамилия.

Здесь 3 основных части. В первой из них хранится информация об открытых пользователем разделах. Во второй части содержится прогресс пользователя. А в третьей имя и фамилия зарегистрированного пользователя. Элементы каждой из частей

разделяются знаком «!», а сами части знаком «\». Обозначение true говорит о том, что соответствующий раздел открыт, а false, что он закрыт. Сами разделы нумеруются последовательно, начиная с первого. Для более понятного объяснения просто расшифруем вышеприведенную строчку.

*Первый раздел открыт!Второй раздел закрыт!Третий раздел закрыт!Четвертый раздел закрыт\Первый раздел прочитан!Тест первого раздела не выполнен!Очков за тест первого раздела 0\Второй раздел не прочитан!Тест второго раздела не выполнен!Очков за тест второго раздела 0\Третий раздел не прочитан!Тест третьего раздела не выполнен!Очков за тест третьего раздела 0\Четвертый раздел не прочитан!Тест четвертого раздела не выполнен!Очков за тест четвертого раздела 0\Имя зарегистрированного пользователя!Фамилия зарегистрированного пользователя*

Все файлы пользователей сохраняются в этом расширении в специально отведенную для этого папку «File_save», располагающуюся в корневом каталоге программы.

Следующий вид файлов: это файлы с расширением .txt. Для основной информации было взято именно это расширение, как наиболее простое, понятное и чаще всего встречающееся. Огромный плюс этого расширения в том, что оно присутствует на всех компьютерах и не требует для подготовки информации или ее прочтения каких либо специальных программ и средств. Для этого будет достаточно обычного блокнота.

Для того, чтобы создать раздел обучения достаточно открыть блокнот, набрать там текст нужного вам раздела и сохранить его под именем, например «section1» или «section2» соответственно для первого и второго разделов, а затем переместить его в корневой каталог программы. Для того чтобы создать другой раздел, все делается аналогично, только названия должны быть соответственно: «start» для начального раздела программы, «program» для информации о программе и «advice» для справочной информации.

Файл информации о прогрессе пользователя не меняется, автоматически заполняется программой и также сохранен в формате .txt под именем «progress». Начальный и не заполненный вариант этого файла имеет вид:

<

*Прогресс пользователя.*

*Открытые разделы:*

*Раздел 1:*

*Раздел 2:*

*Раздел 3:*

*Раздел 4:*

*Имя пользователя:*

*Фамилия пользователя:*

>

Заполненный вариант представляет информацию о достижениях пользователя следующим образом: например

<

*Прогресс пользователя.*

*Открытые разделы: 1, 2, 3;*

*Раздел 1: Прочтен - ДА; Выполнен тест - ДА; Очки за тест - 5;*

*Раздел 2: Прочтен - ДА; Выполнен тест - ДА; Очки за тест - 5;*

*Раздел 3: Прочтен - ДА; Выполнен тест - НЕТ; Очки за тест - 0;*

*Раздел 4: Прочтен - НЕТ; Выполнен тест - НЕТ; Очки за тест - 0;*

*Имя пользователя: Дмитрий;*

*Фамилия пользователя: Анисимов;*

>

Файл прогресса также содержится в корневом каталоге программы.

Следующим и последним расширением, использующимся в программе, является расширение .xml. Как уже упоминалось, оно используется для создания файлов тестирования. Файлы тестов содержатся в папке «Tasks». В нашем случае на каждый раздел запланировано по 4 теста. Программа заходит в папку с тестами и случайным образом выбирает тест для открытого раздела. Затем при нажатии на кнопку «Получить задание» пользователю предоставляется выбранный тест. Название тестов должно начинаться со слова test, далее должен идти номер раздела, к которому этот тест относится, а затем уже сам номер теста. Например, название «test32.xml» означает, что это тест номер 2 к 3 разделу. Теперь разберем саму структуру создания тестов.

<

```
<?xml version="1.0" encoding="Windows-1251"?>
<test>
<head>Заголовок теста</head>
<description>Описание теста</description>
<qw>
<q text="Вопрос номер 1">
<a right="no">Неправильный ответ</a>
<a right="yes">Правильный ответ</a>
<a right="no">Неправильный ответ</a>
</q>
<q text="Вопрос номер 2">
<a right="no">Неправильный ответ</a>
<a right="no">Неправильный ответ</a>
<a right="yes">Правильный ответ</a>
</q>
<q text="Вопрос номер 3">
<a right="no">Неправильный ответ</a>
<a right="yes">Правильный ответ</a>
<a right="no">Неправильный ответ</a>
</q>
```

```
<q text="Вопрос номер 4">
<a right="yes">Правильный ответ</a>
<a right="no">Неправильный ответ</a>
<a right="no">Неправильный ответ</a>
</q>
<q text="Вопрос номер 5">
<a right="no">Неправильный ответ</a>
<a right="no">Неправильный ответ</a>
<a right="yes">Правильный ответ</a>
</q>
</qw>
<levels>
<level score="5" text="На все вопросы Вы ответили правильно. Оценка - 5. Тест зачтен."/>
<level score="4" text="Вы не ответили на 1 вопрос. Оценка - 4. Тест зачтен."/>
<level score="3" text="Вы не ответили на 2 вопроса. Оценка - 3. Тест зачтен."/>
<level score="2" text="Вы не ответили на 3 вопроса. Оценка - 2. Тест не зачтен."/>
<level score="1" text="Вы  ответили правильно только на 1 вопрос. Оценка - 1. Тест не зачтен."/>
<level score="0" text="На все вопросы Вы ответили не правильно. Оценка - 0. Тест не зачтен. "/>
</levels>
</test>
>
```

В данном примере приведен текст файла testExample, также расположенного в папке «Tasks». Этот файл служит начальным вариантом для создания теста. Здесь

Тег <head> обозначает начало заголовка теста, а тег </head> конец этого заголовка. Заголовок теста будет выводиться в самой верхней части программы.

Тег <description> обозначает соответственно начало описания теста, а тег </description> конец этого описания. Описание теста выводится сразу же после нажатия пользователем кнопки «Получить задание».

Тег <qw> обозначает начало перечисления вопросов, а тег </qw> конец этого перечисления.

Тег <q> обозначает начало нового вопроса с ответами, а тег </q> его конец. Для того чтобы написать вопрос надо заполнить атрибут «text» тега <q>.

Тег <a> обозначает начало ответа для вопроса, а тег </a> его конец. Для того чтобы задать правильность или неправильность ответа надо заполнить атрибут «right» тега <a> соответственно значениями «yes» или «no».

Должна присутствовать определенная вложенность тегов, а именно <qw><q><a></a></q></qw>. Также тест testExample рассчитан только на 5 вопросов. Для того чтобы уменьшить или увеличить количество вопросов, потребуется изменять директиву <levels></levels> этого теста.

Приведем пример готового вопроса:

```
<
```

```
    </q>
    <q text="Что обеспечивает двойная спираль в молекуле ДНК?">
    <a right="yes">Возможность удвоения ДНК при размножении клетки.</a>
    <a right="no">Возможность взаимодействия с другими молекулами.</a>
    <a right="no">Возможность синтеза азотистых оснований.</a>
    </q>
>
```

Тег <levels> обозначает начало перечисления расшифровки результатов, а тег </levels> ее конец.

Тег <level/> обозначает расшифровку соответствующего результата за тест. Атрибут «score» этого тега означает сам результат. В нашем случае: числа от 0 до 5. 0 – ни одного правильного ответа, 1 – один правильный ответ, 2 – два правильных ответа, 3 – три правильных ответа, 4 – четыре правильных ответа, 5 – пять правильных ответов. В атрибут «text» тега <level/> записывается сама расшифровка. Например, в нашем случае, для значения score=4, она будет выглядеть *text="Вы не ответили на 1 вопрос. Оценка - 4. Тест зачтен."*.

Опять требуется следующая вложенность тегов <levels><level/></levels>. Общая вложенность тегов такова:

```
<test>
<head></head>
<description></description>
<qw><q><a></a></q></qw>
<levels><level/></levels>
</test>
```

Здесь тег <test> обозначает начало описания теста, а тег </test> конец этого описания. Теперь приведем пример полностью заполненного теста.

```
<
    <?xml version="1.0" encoding="Windows-1251"?>
    <test>
    <head>Вопросы к первому разделу обучения</head>
    <description>Сейчас Вам будут предложены вопросы по пройденному
    материалу из раздела:
    "История ДНК-компьютеров и ДНК-вычислений".
    Из нескольких вариантов ответа Вам следует выбрать
    правильный.</description>
    <qw>
    <q text="Что такое ДНК-компьютер?">
    <a right="no">Компьютер, сделанный из ДНК.</a>
    <a right="yes">Устройство, использующее молекулярные алгоритмы для
    решения различных задач.</a>
    <a right="no">Компьютерная модель, решающая задачи, используя
    молекулярные ДНК алгоритмы.</a>
    </q>
    <q text="Когда впервые было предложено использовать живые ячейки и
    молекулярные комплексы?">
    <a right="no">Начало 1950 годов.</a>
```

```
<a right="no">Середина 1950 годов.</a>
<a right="yes">Конец 1950 годов.</a>
</q>
<q text="В чем основное преимущество молекулярных вычислений?">
<a right="no">В возможности решения большого числа разнообразных задач одновременно.</a>
<a right="yes">В возможности осуществления массового перебора.</a>
<a right="no">В миниатюрном размере молекул.</a>
</q>
<q text="В каком году впервые появился ДНК-компьютер и кем он был создан?">
<a right="yes">1994 г., Л. Эдлман.</a>
<a right="no">1996 г., Л. Эдлман.</a>
<a right="no">1994 г., Р. Фейнман.</a>
</q>
<q text="Какие проблемы возникали в первом опыте Л. Эдлмана?">
<a right="no">Существовала проблема масштабирования задачи, был запрет на использование ДНК в качестве вычислительных устройств.</a>
<a right="no">Не было подходящей лабораторной техники, требовалась чрезвычайно трудоемкая серия реакций.</a>
<a right="yes">Требовалась чрезвычайно трудоемкая серия реакций, существовала проблема масштабирования задачи.</a>
</q>
</qw>
<levels>
<level score="5" text="На все вопросы Вы ответили правильно. Оценка - 5. Тест зачтен."/>
<level score="4" text="Вы не ответили на 1 вопрос. Оценка - 4. Тест зачтен."/>
<level score="3" text="Вы не ответили на 2 вопроса. Оценка - 3. Тест зачтен."/>
<level score="2" text="Вы не ответили на 3 вопроса. Оценка - 2. Тест не зачтен."/>
<level score="1" text="Вы  ответили правильно только на 1 вопрос. Оценка - 1. Тест не зачтен."/>
<level score="0" text="На все вопросы Вы ответили не правильно. Оценка - 0. Тест не зачтен. "/>
</levels>
</test>
>
```

Остановимся более подробно на блоке заданий. При открытии учебного раздела кнопка «Получить задание» становится активной и при ее нажатии текущий раздел закрывается, убирается вся вспомогательная информация и пользователю выводится описание теста, который он будет выполнять. Тест, как уже упоминалось, выбирается случайным образом. Также появляется кнопка «Отменить задание», при нажатии на которую обучаемый в любой момент может прекратить выполнение теста и вернуться к текущему разделу. При этом информация о пройденном тесте будет потеряна. Если пользователь все-таки закончил тест, то в случае проходного бала он сможет перейти к

следующему разделу, а в случае неудовлетворительной оценки он будет возвращен к текущему разделу.

Если обучаемый, допустим, пройдя 3 раздела решил опять выполнить тест, например, к 1 разделу, но сделал это неудовлетворительно, то доступ к уже открытым им ранее разделам опять будет закрыт, пока он ни выполнит текущий тест на положительную оценку. При этом информация о тестах для открытых им ранее разделов сохранится в его прогрессе.

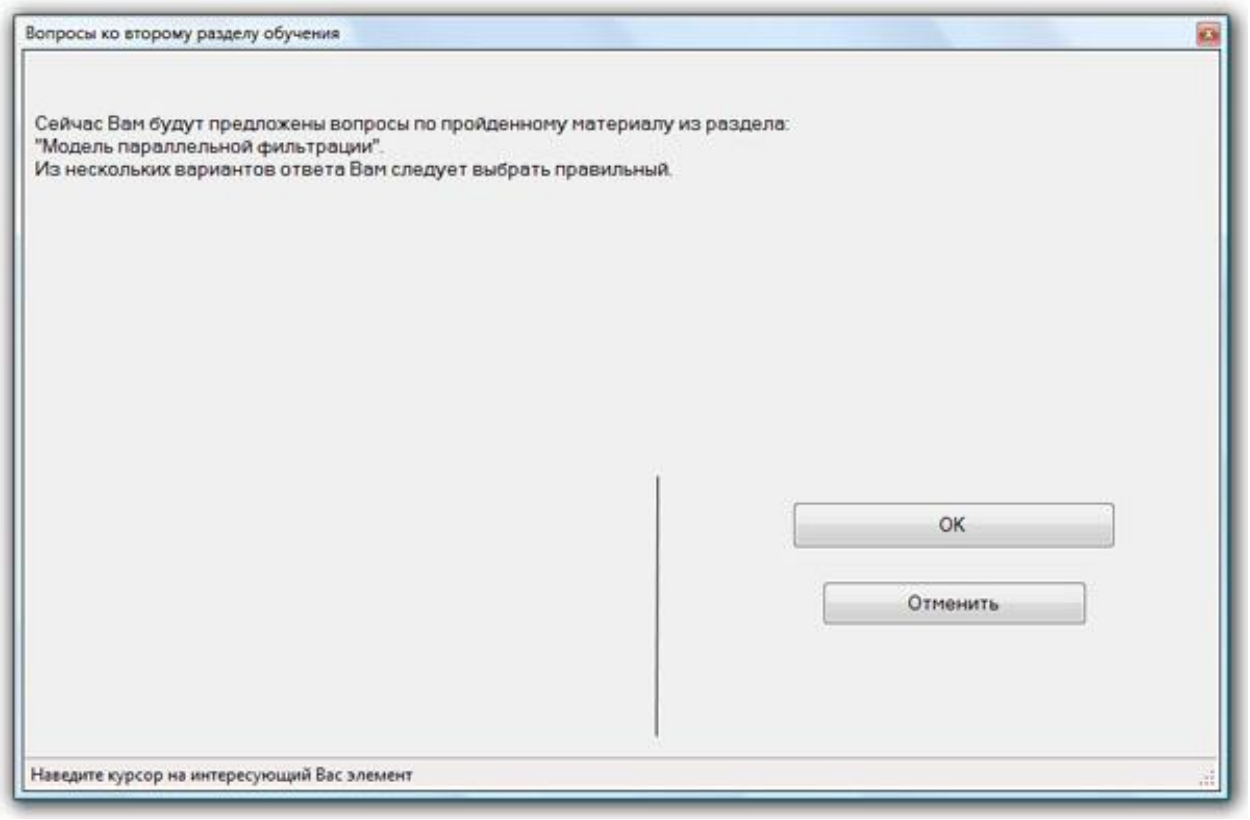


Рис. 12. Начало выполнения теста.

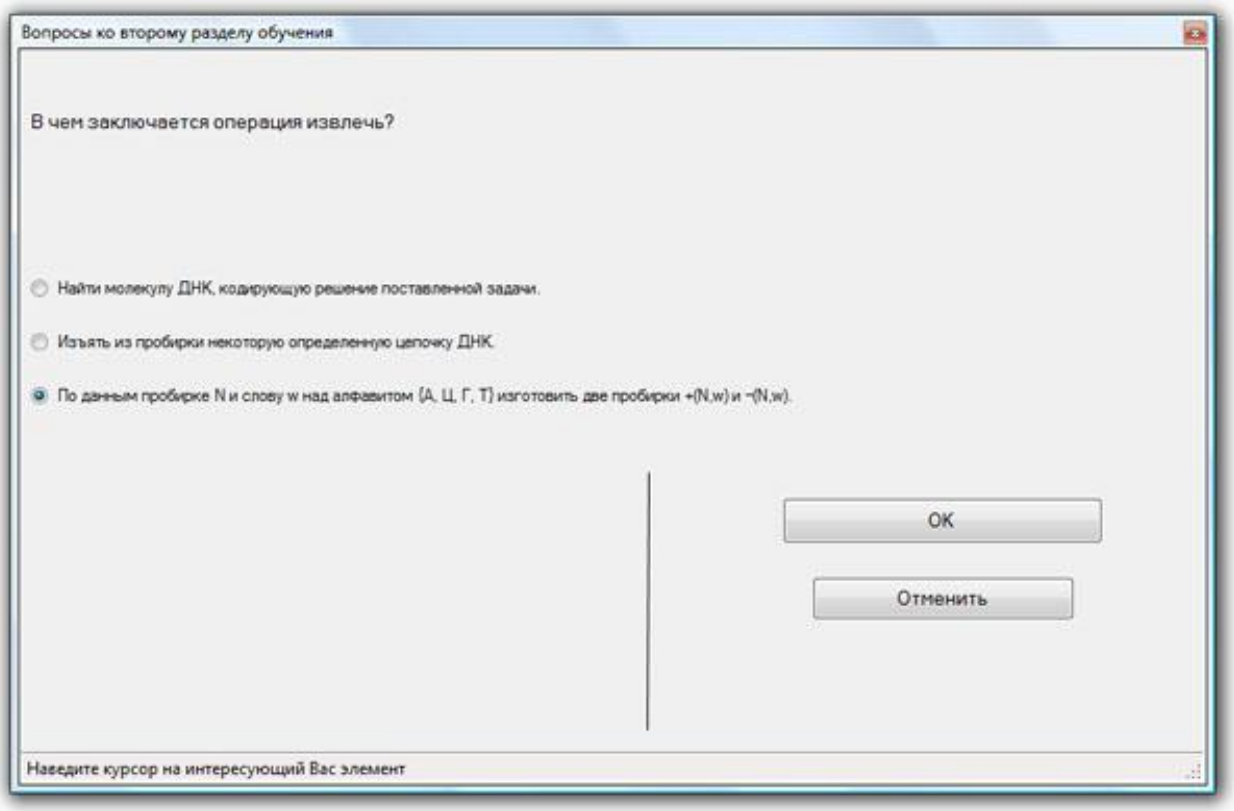


Рис. 13. Процесс выполнения теста.

В конце стоит упомянуть, что все фотографии к разделам хранятся в папках с соответствующими названиями, например, «Section2» для второго раздела обучения. Фотографии должны иметь расширение .jpg и любой размер, т.к. программа, найдя нужную ей фотографию, автоматически ее преобразует к требуемому размеру.

## ✓ Преимущества комплекса DNA-Learning

Приведем некоторые важные особенности данного комплекса:

1. Открытый исходный код.
2. Простота использования.
3. Возможность работы с несколькими пользователями одновременно.
4. Краткое и понятное изложение информации.
5. Просмотр фотографий раздела в реальном времени.
6. Увеличение фотографий.
7. Широкие возможности тестирования пользователя.
8. Учет прогресса и достижений обучаемого.
9. Все файлы сохраняются в простом и доступном на любом компьютере текстовом формате .txt
10. Возможность добавления комментариев и редактирования материала под свои нужды.
11. Подробная справка по всем объектам и элементам программы.
12. Маленький размер памяти, требуемый для хранения программы.
13. Высокая скорость работы.
14. Поддерживает операционные системы Windows Vista, Windows 7 и Windows XP.
15. Подробно прокомментированный код программы.

## ✓ Краткое описание элементов программной реализации системы

Пространство имен программы `namespace DNA_Learning` содержит следующие основные функции:

1. `OpenDocument()`
   Открывает сохраненный пользователем документ, загружает оттуда все данные, выводит список открытых разделов и название открытого документа.
2. `SaveDocument()`
   Сохраняет документ и выводит название сохраненного документа.
3. `SaveDocumentAs()`
   Сохраняет текущий документ под другим именем, заданным пользователем и меняет название текущего документа на новое.
4. `printDocument()`
   Выводит данные о достижениях пользователя на печать.
5. `OpenSection()`
   Открывает требуемый пользователю раздел программы.
6. `FillListBox()`
   Заполняет список фотографий текущего раздела. Если фотографий нет, то блок фотографий отключается.
7. `showHead()`
   Считывает и выводит заголовок теста.
8. `getQw()`
   Считывает вопросы теста и ответы к ним.
9. `showQw()`
   Выводит вопрос и варианты ответа к тесту.
10. `showLevel()`
    Запоминает результаты пройденного пользователем теста и выводит ему эти результаты. В зависимоти от оценки либо открывает доступ к следующему разделу, либо возвращает текущий раздел.
11. `Questions()`
    Определяет номер раздела и выводит соответствующее ему задание.

Все переменные и их использование подробно прокомментировано в коде программы. Программа написана в среде Microsoft Visual Studio 2008 на языке программирования C#.

**Литература:**

**Книги:**


1. Г. Паун, Г.Розенберг, А. Саломаа, ДНК-компьютер Новая парадигма вычислений, Издательство «Мир», Москва, 2004, 527 стр.

2. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн, Алгоритмы Построение и Анализ, Издательский дом «Вильямс», Москва, 2005, 1293 стр.

3. В. И. Игошин, Математическая логика и теория алгоритмов, Издательский центр «Академия», Москва, 2008, 448 стр.

4. В. П. Дьяконов, В. В. Круглов, MatLab 6.5 SP1/7/7 SP1/7 SP2 + Simulink 5/6. Инструменты искусственного интеллекта и биоинформатики, Солон-Пресс, Москва, 2006, 456 стр.

5. Г. Шилдт, Полный справочник по C#, Издательский дом «Вильямс», Москва, 2009, 750 стр.

6. A. Martyn, Theoretical and Experimental DNA Computation, Springer, Berlin, 2005, 173 p.


**Статьи:**


1. Г. Г. Малинецкий, С. А. Науменко, Вычисления на ДНК. Эксперименты. Модели. Алгоритмы. Инструментальные средства. Препринт ИПМ им. М. В. Келдыша РАН, № 29, Москва, 2005.

2. И.Л. Братчиков, Конспект лекций «Математическая логика и теория алгоритмов», 2008 , 4 семестр, 72 ч.

3. D. Boneh, C. Dunworth, R. J. Lipton, J. Sgall. On the computational power of DNA. Discrete Appl. Math., 71 (1996), 79-94.


**Ссылки в Интернет:**


1. http://www.dnacomputer.us/ - общая статья о ДНК-компьютерах.
2. http://www.casi.net/D.BioInformatics1/D.Fall2000ClassPage/DC2/dc2.htm - история ДНК-компьютеров.
3. http://ru.wikipedia.org/wiki/ДНК-компьютер - краткое описание ДНК-компьютера.
4. http://ru.wikipedia.org/wiki/ДНК - описание строения дезоксирибонуклеиновой кислоты.