*Юрай Громкович,*
*Аблаев Фарид Мансурович*

# АЛГОРИТМЫ И СЛУЧАЙНОСТЬ УРОК 5. ОНЛАЙН АЛГОРИТМЫ И КОНКУРЕНТНЫЙ АНАЛИЗ

В этой статье мы рассмотрим класс задач, получивший название онлайн проблемы (online problems), рассмотрим алгоритмы их решения – онлайн алгоритмы (online algorithms). Исследования таких задач и разработки алгоритмов для них – интенсивно развивающаяся область современной Информатики [1].

Итак, что же это такое онлайн задача? Вычислительные задачи, с которыми мы имеем дело на уроках математики и информатики, традиционно выглядят следующим образом: известна постановка задачи, известны все исходные данные, требуется получить ответ, а для этого требуется разработать алгоритм решения этой задачи.


*Онлайн задачи — это задачи, решение которых нужно организовать, не зная заранее всех входных данных, получая решение по мере появления входных данных...*

*Онлайн задачи* – это задачи, решение которых нужно организовать, не зная заранее всех входных данных, получая решение по мере появления входных данных. Алгоритмы, которые разрабатываются при таких условиях называются алгоритмами, работающими в реальном времени, или онлайн алгоритмами (online algorithms). Например, системная программа, контролирующая работу операционной системы, должна производить некоторые действия (принимать решения, записывать что-то на выход) до того, как поступят все данные. Другой пример организация системы диспетчерского центра службы скорой помощи. Входными данными в этом случае является информация о том, сколько будет вызовов, в какое время и где они произойдут. Задача диспетчеров состоит в том, чтобы как можно быстрее машина скорой помощи добралась до места происшествия. Если бы им удалось узнать все входные данные о возможных происшествиях заранее, они смогли бы решить задачу наиболее оптимальным образом, сократив время ожидания до минимума.

В реальных условиях неопределенности перед ними стоит важная задача разработки такой стратегии (алгоритма) поведения, ко-

торая позволит реагировать на происшествия максимально оперативно. Цель-максимум – приблизиться к оптимальному решению, то есть реагировать на новые события так же быстро, как если бы они знали о них заранее.

Мы с вами посмотрим, чем может помочь математика в решении таких задач: есть ли методы, позволяющие строить хорошие стратегии, как для этого используются вероятностные модели.

Рассмотрим еще один пример и подходы к его решению, чтобы получить интуитивное понимание природы онлайн проблем.

Допустим, что вы решили поехать на горнолыжный курорт (например, в Сочи) на 20 дней и собираетесь каждый солнечный день использовать для катания на лыжах. Не секрет, что горные лыжи нельзя отнести к бюджетным видам спорта. Допустим, у вас нет своих собственных лыж. Тогда у вас 2 варианта действий:

– арендовать лыжи (стоимость аренды – 300 руб. за один день проката);

– купить лыжи (минимальная цена подростковых горных лыж 3000 руб.).

Одно важное условие: вы не доверяете долгосрочным прогнозам погоды (особенно в горах) наверняка. Какая погода будет в конкретный день, вы можете более-менее достоверно узнать только утром этого дня. Таким образом, вы не знаете заранее, сколько дней можно будет кататься на лыжах.

Если вы купите лыжи в первый же день, а на следующий день погода испортится до конца вашего тура, вы потратите много денег. С другой стороны, если вы решите каждый день брать лыжи на прокат, то за 20 дней вы потратите стоимость двух пар лыж.

Что же вам делать?

Ваша задача – минимизировать количество денег, потраченных на аренду лыж. Очевидно, что есть смысл покупать лыжи, если вы будете кататься больше 10 дней, иначе ваши затраты будут избыточными.

В целом, создание стратегии похоже на игру с противником, способным контролировать погоду: противник выбирает погоду в конкретный день, вы реагируете на его выбор согласно вашей стратегии. Наша задача – придумать такую стратегию, которая будет давать нам максимальную экономию денег вне зависимости от действий противника (для любой его стратегии). При этом противник очень силен: он не только управляет входными данными, но и знает нашу стратегию поведения, и может создавать «неудобные» входные данные для нашего алгоритма.

Прежде чем читать дальше выполните небольшое упражнение — попробуйте придумать стратегию, которая позволит вам сэкономить деньги.

А теперь мы предложим алгоритм действий противника, чтобы вы могли провести игру с ним согласно вашей стратегии.

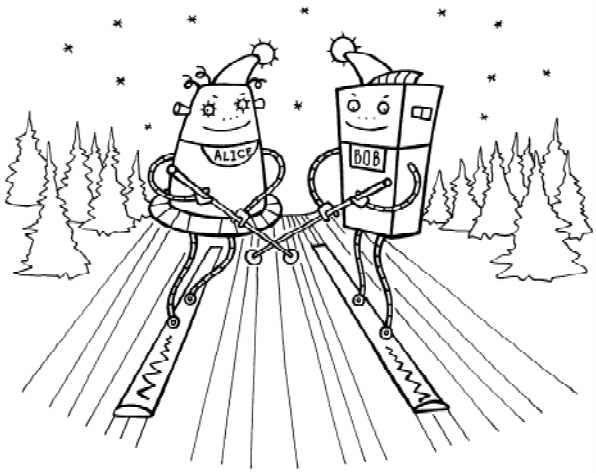
Итак, противник будет действовать следующим образом:

```
while (вы не купили лыжи и ваш отпуск не закончился)
output «Хорошая погода» ;
while (ваш отпуск не закончился)
output «Плохая погода» ;
end.
```

Сколько денег на аренду лыж потратили вы?

Посмотрим, какие случаи могут происходить:

– если вы решитесь на покупку лыж в первый день с хорошей погодой, тогда вы потратите 3000 рублей;




*Ваша задача — минимизировать количество денег, потраченных на аренду лыж.*

– если вы решитесь на покупку лыж в $j$-ый день с хорошей погодой $2 \le j < 10$, тогда вы заплатите $(j-1)300 + 3000$ рублей.

Как бы мы ни действовали, противник может создать такую ситуацию, когда мы потратим не меньше 3000 рублей. Таким образом, на аренде лыж сэкономить не получится, лучше уж сразу купить лыжи (это будет наиболее выгодный вариант).

Вы можете возразить, что нет такого противника, способного управлять погодой, и что может случиться, что солнечных дней будет совсем немного, и стратегия подневной аренды даст максимальную экономию. Поясним этот момент. Эффективность нашей стратегии определяется ее эффективностью во всех возможных случаях, в том числе и в худших для нас случаях. Они ведь тоже могут произойти, а значит, нам нужно предусмотреть наши действия и в этих случаях тоже. Мы рассматриваем противника как модель создания таких худших случаев, чтобы проверить эффективность нашего алгоритма. Для заданного алгоритма мы с помощью противника порождаем такой набор входных данных, на котором заданный алгоритм будет неэффективен.

В чем же причина такой неэффективности наших стратегий (алгоритмов)? Ясно, что если бы мы знали все входные данные о погоде, то мы бы использовали оптимальное решение, что позволило бы нам потратить минимально возможное количество денег. Выходит, что мы платим за незнание будущего. Можем ли мы что-то предпринять в этой ситуации, или все наши попытки тщетны? Давайте попробуем разобраться.

Как мы уже говорили раньше алгоритмы для решения онлайн проблем называются онлайн алгоритмами. Заметим, что для одной и той же задачи можно построить несколько онлайн алгоритмов.

Рассмотрим некоторую онлайн проблему $P$ и некоторый онлайн алгоритм $A$ решения этой задачи. Набор входных данных обозначим $I = (x_1, x_2, ..., x_n)$. В случае задачи аренды лыж $x_i$ могут принимать значения «солнечный день» или «несолнечный день». Сократим обозначения до «с» и «н». Пример входного набора для случая, когда отпуск длится 10 дней: $I$ = (с, н, с, с, с, н, с, с, с, н). На каждое значение входного набора алгоритм должен выдавать ответ $y_i$. Для задачи аренды лыж каждый день мы должны давать ответ: арендовать лыжи, купить лыжи или ничего не делать. Будем обозначать «к» ситуацию, когда мы купили лыжи, «а» – арендовали лыжи, «н» – ничего не делали. Пример выходного набора $O = (y_1, y_2, ..., y_n)$ для заданного ранее входного набора: O = (а, н, к, н, н, н, н, н, н, н). Выходной набор определяется выбранной стратегией. Приведем еще один пример выходного набора, когда мы всегда арендуем лыжи: O = (а, н, а, а, а, н, а, а, а, н).

Для пары значений входного набора и соответствующего ему выходного набора определим *функцию стоимости* $cost(I, O)$, задающую неотрицательное вещественное число. Для задачи аренды лыж мы определим функцию стоимости следующим образом: просмотрим весь выходной набор и для каждого значения «а» возьмем число 300, для значения «к» возьмем число 3000, для каждого значения «н» возьмем число 0; просуммируем получившиеся числа. Результат будет отражать сумму наших затрат на аренду лыж в течение отпуска.

Для каждого входного набора будем называть *оптимальным* тот выходной набор, на котором достигается минимум функции стоимости. Будем обозначать его $Opt(I)$.

**Определение 1.** Рассмотрим некоторую онлайн проблему $P$. Множество входных наборов проблемы $P$ будем обозначать $\mathcal{I}$. Для конкретного входного набора $I \in \mathcal{I}$, $I = (x_1, ..., x_n)$ некоторый онлайн алгоритм $A$ вычисляет выходной набор $O = (y_1, y_2, ..., y_n)$, где $y_i = f_i(x_1, ..., x_i, y_1, ..., y_i)$, $f_i$ – некоторый набор функций, определяемый алгоритмом $A$, $i \in \{1, n\}$. Решение $O$, созданное алгоритмом $A$ на входном наборе $I$, будем обозначать $A(I)$, стоимость этого решения – $cost(A(I))$. Наша задача – минимизировать стоимость. Такая задача называется *задачей онлайн минимизации*.

**Определение 2.** Рассмотрим некоторую онлайн проблему $P$ и онлайн алгоритм $A$, решающий эту проблему. Алгоритм $A$ бу-

дем называть *c-конкурентным*, если существует неотрицательная константа $\alpha$ такая, что для любого $I \in \mathcal{I}$ справедливо соотношение:

$$cost(A(I)) \le c \cdot cost(Opt(I)) + \alpha,$$

где $Opt$ – это оптимальный алгоритм для этой задачи, то есть алгоритм, порождающий оптимальный выходной набор. Значение c мы будем также называть *конкурентным отношением* алгоритма $A$.

Если $\alpha = 0$, то $A$ будем называть *строго конкурентным*.

Алгоритм $A$ – *оптимальный алгоритм*, если он строго 1-конкурентен.

Конкурентное отношение алгоритма $A$ на входном наборе $I$ мы также будем обозначать $comp(A(I))$. Отметим, что наличие константы $\alpha$ позволяет алгоритмам, стоимость решений которых отличается от стоимости решений оптимального алгоритма лишь на константу, становиться 1-конкурентными.

Вернемся к оптимальному алгоритму *Opt*. Это алгоритм, который заранее знает весь входной набор. В действительности его существование чисто гипотетическое. Алгоритм *Opt* обладает мощной информацией, которой не обладает $A$, и это та дилемма, с которой мы встречаемся при построении онлайн алгоритмов. Говоря другими словами: «Конкурентное отношение сообщает нам, сколько мы должны заплатить за незнание будущего».

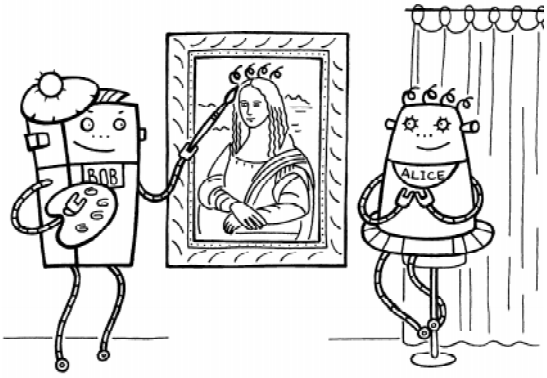
**Задача кэширования (Пейджинга).** Одна из очень важных онлайн задач – это задача кэширования или задача пейджинга. Суть ее состоит в следующем.

Физическая память компьютера многослойна. Центральный процессор в первую очередь оперирует своими регистрами, работа с которыми происходит очень быстро, но сама эта память очень дорога. Далее идет кэш процессора, который дешевле и медленнее, далее – оперативная память, следующая память самая дешевая самая медленная – это память, располагающаяся на жестком диске. Мы сфокусируемся на одном уровне иерархии, когда у нас есть медленная физическая память и быстрая, но дорогая память кэша.

Для более эффективного использования памяти она разбивается на страницы (или ячейки, page). Далее процессор и программы оперируют только страницами, а не отдельными байтами. В случае, если пользователь запрашивает некоторую страницу и она уже находится в кэше, то процессор сразу же берет ее из кэша и отдает пользователю. Если же ее в кэше не оказывается, то происходит так называемый «обмен страницами»: процессор выбирает некоторую ячейку из кэша и записывает данные запрошенной страницы в эту ячейку, после чего отдает данные пользователю. Данные же, поверх которых была записана новая страница, пропадают из кэша. Основная задача алгоритма – это определить, поверх какой ячейки стоит записывать новую страницу. Пусть кэш может хранить в себе не более чем $k$ страниц. Обычно кэш заполнен, поэтому можно считать, что в кэше всегда ровно $k$ страниц.

Опишем задачу формально.

**Определение 3.** Рассмотрим последовательность целых чисел, представляющих запросы страниц памяти $I = (x_1, ..., x_n)$, $x_i > 0$. Онлайн алгоритм $A$ обслуживает буфер (содержимое кэша) $K$ целых чисел $B = \{b_1, ..., b_K\}$, где $K$ – фиксированная константа, причем алгоритм $A$ знает значение этой константы. До первого запроса буфер инициализируется следующим значением $B = \{1, ..., K\}$. Если $A$ принимает запрос $x_i \in B$, то тогда выдает ответ $y_i = 0$. Если же $x_i \notin B$, то происходит «обмен страницами»,




*Основная задача алгоритма — это определить, поверх какой ячейки стоит записывать новую страницу...*

и $A$ должен определить заменяемую страницу $b_j$, при этом $B := B \setminus \{b_j\} \cup \{x_i\}$ и $y_i = b_j$. Стоимость решения $(I)$ равно количеству обменов страниц, то есть $cost(A(I)) = |\{y_i : y_i > 0\}|$.

Для этой задачи для любого онлайн алгоритма $A$ можно построить такого соперника $Adv$, который предъявляет худший входной набор, просто запрашивая в каждый момент времени ту страницу, которой нет в кэше. Формально алгоритм формирования такого входного набора можно записать следующим образом

```
output «Запрос страницы с индексом K+1»;
i:= 1;
while (i\leq K-1)
j:= индекс страницы, которую A убрал из кэша
при замене;
output «Запрос страницы с индексом j»;
i:= i + 1;
end.
```

При этом оптимальный алгоритм $Opt$ мог бы обработать следующим образом такой набор. Он знает всю последовательность запросов. И первую запрошенную страницу заменит на ту, которую будут запрашивать максимально поздно так, чтобы предыдущие оставались в кэше. В худшем случае соперник будет запрашивать все страницы подряд и алгоритму $Opt$ понадобится заменять страницы каждые $K$ запросов.

Таким образом справедлива следующая теорема.

**Теорема 1.** *Для любого онлайн алгоритма $A$ существует такой входной набор $I$, что выполняется следующее неравенство:*

$$comp(A(I)) = \frac{cost(A(I))}{cost(Opt(I))} \geq K,$$

*где $Opt$ – оптимальный алгоритм.*

**Упражнение 1.** Рассмотрим пример $I = (3, 107, 30, 1201, 73, 107, 30)$. Это означает, что пользователь последовательно запросил страницы с номерами 3, 107, 30, 1201, 73, 107, 30. Как видим, в этом случае страницы 30 и 107 запрашивались дважды, и было бы неправильно удалить их из кэша после первого использования.

Нам нужно минимизировать стоимость. Для начала определим стоимость каждого возможного решения. Поскольку время, необходимое для перемещения страницы из основной памяти в кэш, намного больше, чем для доступа к данным в кэше, можно считать стоимость так:

– стоимость доступа к кэшу равна 0;

– стоимость перемещения страницы в кэш равна 1.

Пусть размер нашего кэша, в котором хранятся страницы 5, 30 и 107, равен 3 (то есть $k = 3$). И нам подается задача $I = 3, 107, 30, 1201, 73, 107, 30$. Покажем, что следующее решение, имеющее стоимость 3, является оптимальным. Сначала компьютер переместит страницу 5 в основную память и на её место запишет страницу 3. Будем обозначать такую операцию как

$$5 \leftrightarrow 3.$$

Потом запрашиваются страницы 107 и 30, которые и так находятся в кэш-памяти. Обработав эти страницы, компьютер перемещает в кэш страницу 1201, заменяя страницу 3. Страницы 30 и 107 остаются, так как компьютер знает, что они понадобятся позже. На пятом шаге страница 1201 из кэша заменяется страницей 73. И последние запрашиваемые страницы 30 и 107 обрабатывается напрямую из кэша . Это решение можно также записать в виде

$$5 \leftrightarrow 3, \bullet, \bullet, 3 \leftrightarrow 1201, 1201 \leftrightarrow 73, \bullet, \bullet.$$

Символ $\bullet$ означает, что кэш и основная память не взаимодействовали на этом шаге, а $a \leftrightarrow b$ означает замену страницы $a$ из кэша на страницу $b$ и запись $a$ в память.

Покажем, что это решение является оптимальным. Сначала страницы 3, 1201 и 73 находятся не в кэш памяти, но если требуется обработать страницы $I = 3, 107, 30, 1201, 73, 107, 30$ , то эти страницы должны быть загружены в кэш. Это значит, что количество операций не может быть меньше трех.

**Упражнение 2.** Найти оптимальные решения для задач:

1. $k$ = 3, в кэше находятся 1, 2, 3 и $I = 7, 9, 3, 2, 114, 8, 7$.

2. $k$ = 5, в кэше находятся 1, 101, 1001, 1002, 9 и $I = 102, 7, 5, 1001, 101, 3, 8, 1, 1002$.

Заметим, что когда нужно записать новую страницу в кэш, есть $k$ возможностей выбрать в кэше страницу для замены. И с ростом количества шагов количество разных возможных решений растет очень быстро, поэтому найти среди них оптимальное очень сложно. Но все же это возможно. Если рассматривать эту задачу как онлайн задачу, то ситуация меняется. Требования на обработку страниц приходят уже после того, как мы поместили туда предыдущую, причем вместо какой-то другой страницы. И тогда какую бы стратегию мы не использовали, противник будет запрашивать именно ту страницу, которую мы только что убрали из кэша. Ясно, что тогда стоимость будет максимально возможной и равной длине задачи.

Покажем, как это выглядит на примере. Пусть $k = 3$ и в кэше хранятся сраницы 1, 2 и 3. Сначала наш противник запрашивает станицу 4. Тогда какую-то страницу из кэша надо заменить этой страницой. Предположим, что заменяют страницу 2. Тогда на следующем шаге противник запрашивает страницу 2, которой в уже нет в кэше. Пусть тогда компьютер сделает $4 \leftrightarrow 2$. Тогда на следующем шаге противник запросит страницу 4. Если же наш алгоритм решит сделать $1 \leftrightarrow 4$, то противник запросит страницу 1. Таким образом, противник дал на задачу

$$4, 2, 4, 1$$

и вынудил наш алгоритм принять решение

$$2 \leftrightarrow 4, 4 \leftrightarrow 2, 1 \leftrightarrow 4, 4 \leftrightarrow 1,$$

стоимость которого максимальна и равна 4. Оптимальноое же решение этой задачи

$$3 \leftrightarrow 4, \bullet, \bullet, \bullet$$

имеет стоимость 1. Но это оптимальное решение можно найти, только если знать всю задачу заранее.

**Литература**

1. *Hromkovic J.* Algorithmic adventures. From Knowledge to Magic. Springer, 2009.

***Юрай Громкович,***
***Professor of Computer Science, Swiss Federal Institute of Technology, Zürich,***

***Аблаев Фарид Мансурович,***
***доктор физико-математических наук, профессор, заведующий кафедрой теоретической киберенетики Казанского федерального университета.***