

План лекции

- 1 Введение
- 2 Задача кэширования
- 3 Задача о покрытии множествами

с/к “Эффективные алгоритмы”

Лекция 12: Алгоритмы, обрабатывающие вход по мере поступления

А. Куликов

Computer Science клуб при ПОМИ
<http://logic.pdmi.ras.ru/~infclub/>

План лекции

- 1 Введение
- 2 Задача кэширования
- 3 Задача о покрытии множествами

Введение

Введение

- Все рассмотренные нами до сегодняшнего дня алгоритмы получали свои входные данные сразу.
- На практике же существуют задачи, которые нужно решать, не зная заранее все входные данные.
- Например, системная программа, контролирующая работу операционной системы, должна производить некоторые действия (принимать решения, записывать что-то на выход) до того, как поступят все данные.
- Такие алгоритмы называются алгоритмами, работающими в реальном времени, или **online-алгоритмами**.

Оценка эффективности

Оценка эффективности

- Как правило, online-алгоритм получает последовательность запросов.
- По каждому из запросов он должен предоставить некоторый сервис до того, как получит следующий запрос.
- Обычно есть несколько возможностей предоставить сервис, каждой из которых сопоставлена некоторая стоимость.

Оценка эффективности

Определение

- Online алгоритм A называется c -оптимальным (c -competitive), если для любой последовательности запросов r и любого алгоритма B , которому вся последовательность запросов дается сразу,

$$\text{cost}_A(r) \leq c \cdot \text{cost}_B(r) + c_1.$$

- Коэффициентом оптимальности (competitiveness coefficient) c_A алгоритма A называется инфимум таких c , что A c -оптимален.

Аренда лыж

Аренда лыж

- Представьте себя на горнолыжном курорте.
- Вы решили кататься, пока позволяет погода, поэтому заранее не знаете, сколько именно (вы не верите прогнозам погоды).
- У вас нет лыж, но есть две возможности: арендовать лыжи за 10 руб. в сутки или купить лыжи за 150 руб.
- Если вы купите лыжи в первый же день, а на следующий день все растает, вы потратите много денег.
- С другой стороны, если вы решите каждый день брать лыжи на прокат, то за два месяца вы потратите стоимость четырех пар лыж.
- Что же вам делать?

Поиск припаркованной машины

Поиск припаркованной машины

- Рассеянный профессор, выйдя из института, осознал, что не помнит, где припарковал свою машину.
- Будем считать, что здание института круглое и машина припаркована рядом с ним.
- Что же делать профессору?

Online-алгоритм

Online-алгоритм

- Заметим, что online-алгоритм для задачи аренды лыж полностью задается днем N , в который нужно купить лыжи (после этого платить не нужно).
- Итак, какое же N оптимально?
- Если N мало, мы рискуем купить лыжи и уехать с ними на следующий день домой.
- Если же N велико, мы рискуем прокатать до покупки несколько стоимостей лыж.

2-оптимальный online-алгоритм

2-оптимальный online-алгоритм

- 2-оптимальный алгоритм: брать лыжи в прокат первые 14 дней, а на 15-й день купить их.
- Легко видеть, что данный алгоритм 2-оптимален.

Online-алгоритм для поиска припаркованной машины

Online-алгоритм для поиска припаркованной машины

- Рассмотрим такой алгоритм: профессор сходит направо на расстояние 1, потом вернется и сходит налево на расстояние 2, потом опять вернется и сходит направо на расстояние 4, и так далее до тех пор, пока не найдется машина.
- Покажем, что данный алгоритм 9-оптимален.
- Неприятнее всего профессору тогда, когда он не дошел маленького ϵ до своей машины на одной из итераций.
- В такой ситуации ему придется вернуться назад, сходить в противоположную сторону и только потом найти машину.
- Пусть расстояние до машины равно $2^{i+1} + \epsilon$. Тогда ему придется пройти

$$2 \cdot (1 + 2 + \dots + 2^{i+2}) + 2^{i+1} + \epsilon = 2 \cdot (2^{i+3} - 1) + 2^{i+1} + \epsilon = 9 \cdot 2^{i+1} + \epsilon - 2.$$

План лекции

- 1 Введение
- 2 Задача кэширования
- 3 Задача о покрытии множествами

Задача кэширования (paging problem)

Задача кэширования (paging problem)

- У компьютера есть два вида памяти: дисковая, большая по объему, но медленная, и оперативная, быстрая, но меньше по объему.
- Все операции над данными (в частности, запуск программ) производятся в оперативной памяти; дисковая память используется лишь для их хранения.
- Если обращения к одному и тому же месту дисковой памяти повторяются, можно сэкономить время на том, что выделить в оперативной памяти определенное место (кэш), в которой сохранять считанные данные.
- Поскольку объем кэша ограничен, периодически придется стирать из него данные (записывать на их место другие); качество алгоритма определяется тем, насколько он сможет предугадать, какие данные еще понадобятся (и их не следует стирать), а какие — нет.

Формальная постановка задачи

Формальная постановка задачи

- Вся память (и дисковая, и оперативная) разбита на блоки равной длины.
- Блоки пронумерованы, причем кэш состоит из k блоков, а диск — из t блоков.
- На очередном шаге к нам поступает запрос на какой-то блок с диска; если этот блок в кэше уже имеется, нам достаточно сообщить его номер в кэше; если нет, нам надо предварительно решить, в какое место кэша записать этот блок и сделать это, считав блок с диска.
- Время работы алгоритма измеряется количеством считываний с диска.
- Будем считать, что в начальный момент времени кэш пуст.

Известные эвристики

Известные эвристики

- Когда приходит запрос на блок, которого нет в кэше, online-алгоритму приходится стереть какой-нибудь блок из кэша и на его место записать запрошенный блок.
- Стандартные эвристики выбора такого блока:
 - Least Recently Used, LRU: стереть блок, последний запрос на которой был раньше всего.
 - First-in, First-out, FIFO: стереть блок, находящийся в кэше дольше всего.
 - Least Frequently Used, LFU: стереть блок, который запрашивался реже всего.

Оптимальный offline-алгоритм

Алгоритм MIN

При запросе блока, которого нет в кэше, стереть из кэша тот блок, запрос на который будет в будущем позже всего.

Упражнения

Пусть диск состоит из $t = k + 1$ блока.

- Показать, что для любого детерминированного offline-алгоритма A для задачи кэширования существует последовательность из N запросов r , такая что $\text{cost}_A(r) = N$.
- Показать, что для любой последовательности из N запросов r $\text{cost}_{MIN}(r) = N/k$.

Нижняя оценка для детерминированных алгоритмов

Лемма

Для любого детерминированного алгоритма A для задачи кэширования $c_A \geq k$.

Идеи доказательства

- Мы покажем, что существует offline-алгоритм B и последовательность запросов r со следующими условиями.
- Последовательность делится на периоды, где под периодом понимается максимальная последовательность, содержащая запросы на k различных блоков.
- За каждый период A делает k считываний с диска, а B — всего один.
- Последовательность строится по алгоритму A (каждый следующий запрос всегда можно сделать на блок, которого точно нет в кэше алгоритма A).

Случайные числа

Случайные числа

- Итак, детерминированные online-алгоритмы не особо хороши.
- Можем ли мы улучшить алгоритм, позволив ему пользоваться случайными числами?
- Для этого сначала нужно понять, из какого класса берется **противник** (алгоритм, относительно которого мы оцениваем наш алгоритм).
- Противник выдает всю последовательность сразу? Или же постепенно, смотря на наши ответы?
- Если он смотрит на наши ответы, то сразу ли он сам отвечает на свои запросы?

Модели противников

Модели противников

Будем считать, что все перечисленные ниже типы противников могут знать наш алгоритм, но не знают случайных чисел, которыми мы пользуемся.

Забывчивый противник (oblivious adversary) порождает всю последовательность заранее.

Активный offline-противник (adaptive offline adversary) порождает последовательность запросов, смотря на наши ответы.

Активный online-противник (adaptive online adversary) порождает последовательность запросов, смотря на наши ответы, но обязан сразу выдавать ответы на свои запросы.

Замечание

Мы рассмотрим самую слабую из данных моделей, то есть забывчивого противника.

Забывчивый противник

Забывчивый противник

- Мы будем строить вероятностный алгоритм и оценивать его относительно забывчивого противника.
- Поскольку наш алгоритм вероятностный, мы будем оценивать мат. ожидание количества сделанных им считываний с диска.

Алгоритм Marker

Алгоритм

MARKER

- Каждый блок кэша будем помечать 0 или 1.
- Все время работы разделяется на периоды.
- В начале каждого периода все блоки кэша помечены 0.
- Если приходит запрос на блок, который есть в кэше, помечаем этот блок 1.
- Если же запрошенного блока нет, случайным образом выберем блок из помеченных 0, считаем туда требуемый блок и пометим этот блок 1.
- Если пришел запрос на блок, которого в кэше нет, и все блоки помечены 1, обнулیم все пометки и начнем новый период.

Оценка оптимальности

Лемма

Алгоритм MARKER является $2H_k$ -оптимальным относительно забывчивого противника.

Типы запросов

- Помеченный** — запрос на блок, образ которого находится в кэше и помечен 1.
- Устаревший** — запрос на блок, образ которого находится в кэше и помечен 0 (блок, оставшийся в кэше с предыдущего периода), или на блок, которого в кэше нет, но который там был на предыдущем периоде.
- Чистый** — запрос на блок, которого в кэше нет и не было на предыдущем периоде.

Оценка кол-ва считываний противника

Оценка кол-ва считываний противника

- Обозначения:
 - ▶ l_i — кол-во чистых запросов за i -й период.
 - ▶ $S_{O,i}, S_{M,i}$ — множества блоков в кэшах противника и нашего алгоритма, соответственно.
 - ▶ $d_{I,i}, d_{F,i}$ — значения величины $|S_{O,i} \setminus S_{M,i}|$ в начале и конце i -го периода, соответственно.
- Наш алгоритм обязан подгрузить l_i блоков.
- Значит, противник подгрузит хотя бы $l_i - d_{I,i}$ блоков: он может выиграть у нас только за счет того, что у него в начале периода будут блоки, на которые поступят запросы.
- Это кол-во также составляет не менее $d_{F,i}$: все блоки, лежащие к концу периода в кэше нашего алгоритма, были запрошены; значит, они должны были побывать и в кэше противника; если же кого-то из них там не оказалось, значит, на его место был загружен другой блок; кол-во таких загрузок не менее $d_{F,i}$.

Оценка кол-ва считываний противника (продолжение)

Оценка кол-ва считываний противника

- Итак, количества считываний противника за i -й период составляет не менее

$$\max\{l_i - d_{I,i}, d_{F,i}\} \geq \frac{l_i - d_{I,i} + d_{F,i}}{2}.$$

- Просуммировав по всем периодам, получим нижнюю оценку на общее число загрузок, совершенных противником:

$$\sum_i \frac{l_i - d_{I,i} + d_{F,i}}{2} = \sum_i \frac{l_i}{2} + \frac{d_{F,n}}{2} \geq \frac{L}{2},$$

где $L = \sum_i l_i$.

Оценка кол-ва считываний алгоритма

Оценка кол-ва считываний алгоритма

- Алгоритм подгружает новый блок на каждый чистый запрос.
- На помеченные ничего подгружать не нужно.
- В i -м периоде ровно $k - l_i$ устаревших запросов.
- При устаревшем запросе мы обращаемся к блоку, который на предыдущем периоде был в кэше, но есть ли он там сейчас, мы сказать не можем (он мог быть выгружен).
- Посчитаем вероятность того, что j -й устаревший запрос происходит на блок, которого нет в кэше.
- Вспомним, что для каждого устаревшего блока в начале периода был соответствующий блок.
- Пусть X — множество позиций кэша, в которых в начале периода были блоки из первых $j - 1$ устаревших запросов.
- Ясно, что перед j -м устаревшим запросом все эти позиции помечены 1.

Оценка кол-ва считываний алгоритма (продолжение)

Оценка кол-ва считываний алгоритма

- Каждая позиция из X могла быть помечена 1 либо при нахождении соответствующего устаревшего блока, либо при загрузке туда чистого блока.
- Тогда из множества X (в котором наш блок и лежит) не более l раз производилась случайная выборка.
- Посчитаем вероятность того, что наш блок при этом выживет.

Оценка кол-ва считываний алгоритма (продолжение)

Оценка кол-ва считываний алгоритма

- Пусть $|X| = k - j + 1 = K$.
- Тогда вероятность ему выжить на первом шаге не менее $K/(K - 1)$, на втором — не менее $(K - 1)/(K - 2)$ и т.д.
- Перемножив, получим, что вероятность выжить за все l выборок хотя бы $(K - l)/K$.
- Значит, с вероятностью не более $\frac{l}{K} = \frac{l}{k-j+1}$ алгоритму придется считать блок при j -м устаревшем запросе.
- Тогда мат. ожидание количества считываний при устаревших запросах не превосходит

$$\frac{l_i}{k} + \frac{l_i}{k-1} + \dots + \frac{l_i}{l_i+1} = l_i \cdot (H_k - H_{l_i}).$$

Оценка кол-ва считываний алгоритма (продолжение)

Оценка кол-ва считываний алгоритма

- Таким образом, мат. ожидание общего числа загрузок за i -й период (включая l_i чистых загрузок) будет не более

$$l_i \cdot (1 + H_k - H_{l_i}) \leq l_i \cdot H_k.$$

- Просуммировав по всем периодам, получим, что мат. ожидание общего числа загрузок нашего алгоритма не более $L \cdot H_k$.
- Вспомнив, что наш противник делает хотя бы $L/2$ загрузок, получаем требуемую оценку на оптимальность. □

План лекции

- 1 Введение
- 2 Задача кэширования
- 3 Задача о покрытии множествами

Online-вариант задачи о покрытии множествами

Online-вариант задачи о покрытии множествами

- Пусть дано множество $X = \{1, 2, \dots, n\}$ и множество его подмножеств \mathcal{S} , $|\mathcal{S}| = m$, каждому из которых присвоен вес 1. Считаем, что все подмножества в объединении покрывают X .
- **Покрытием** (cover) называется множество подмножеств \mathcal{S} , в объединении покрывающих X .
- **Online-вариант задачи о покрытии множествами** (online set cover problem) определяется как следующая игра между алгоритмом и его противником:
 - ▶ противник дает алгоритму элементы множества X по одному;
 - ▶ на каждый элемент алгоритм должен ответить множеством из \mathcal{S} , покрывающим этот элемент;
 - ▶ противник дает алгоритму не все X , а некоторое его подмножество $X' \subseteq X$; алгоритм знает заранее X и \mathcal{S} , но не знает X' ;
 - ▶ цель алгоритма — минимизировать количество множеств.

Оценка качества алгоритма

Оценка качества алгоритма

- Эффективность алгоритма будем оценивать относительно активного offline-противника.
- То есть после того, как противник выдал алгоритму все элементы из X' , он сам выдает некоторое решение для X' .
- Мы будем доказывать верхнюю оценку на коэффициент оптимальности алгоритма, то есть показывать, что построенное алгоритмом множество всегда несильно больше оптимального.

Пример применения задачи

Пример применения задачи

- Рассмотрим сеть серверов.
- Есть потенциальное множество клиентов, и каждый сервер способен предоставить некоторый сервис какому-то подмножеству клиентов.
- Запуску каждого сервера присвоена некоторая цена.
- Клиенты присылают запросы последовательно, и отвечать на них нужно сразу.

Основные идеи алгоритма

Основные идеи алгоритма

- Каждому множеству присвоим некоторый вес.
- При приходе очередного элемента веса всех множеств, содержащих данный элемент, некоторым образом пересчитываются.
- Каждое из этих множеств будет выбрано алгоритмом на этом шаге с вероятностью, примерно пропорциональной увеличению его веса.

Более формально

Более формально

- Для каждого множества будем поддерживать вес $w_S > 0$.
- В течение работы алгоритма веса могут только увеличиваться.
- Изначально $w_S = \frac{1}{2m}$ для каждого $S \in \mathcal{S}$.
- Через \mathcal{C} будем обозначать текущее покрытие, а через C — текущее множество покрытых элементов.
- Вес элемента: $w_j = \sum_{S \in \mathcal{S}_j} w_S$, где $\mathcal{S}_j = \{S \in \mathcal{S} : j \in S\}$.
- Потенциал:

$$\Phi = \sum_{j \notin C} n^{2w_j}.$$

Алгоритм

Алгоритм

ONLINE-SET-COVER(j)

- Если $w_j \geq 1$, выйти.
- В противном случае пересчитываем веса:
 - ▶ Пусть k — минимальное натуральное число, для которого $2^k w_j > 1$ (ясно, что $2^k w_j < 2$).
 - ▶ Для каждого $S \in \mathcal{S}_j$ $w_S = 2^k w_S$.
 - ▶ Выбрать из \mathcal{S}_j не более $4 \log n$ множеств в \mathcal{C} , так чтобы значение потенциала не превосходило значения потенциала до пересчета весов.

Анализ количества итераций

Лемма

Количество итераций, на которых происходит пересчет весов, не превосходит $|\mathcal{C}_{\text{opt}}| \cdot (\log m + 2)$.

Доказательство

- Для любого множества S на каждой итерации $w_S \leq 2$, поскольку $w_j \leq 2$ для всех элементов j .
- Пересчет весов происходит только в случае, если $w_j < 1$.
- При пересчете весов вес хотя бы одного из множеств покрытия \mathcal{C}_{opt} умножается на число, не меньшее 2.
- Поскольку изначально вес каждого множества равен $\frac{1}{2m}$, а в конце — не более 2, то каждое множество может участвовать не более чем в $\log(4m)$ увеличениях.

□

Оценка потенциала

Лемма

Всегда найдутся такие $4 \log n$ множеств при пересчете весов, при выборе которых потенциал не увеличится.

Доказательство

- Для каждого $S \in \mathcal{S}_j$ через w_S и $w_S + \delta_S$ будем обозначать вес S до и после пересчета.
- Пусть $\delta_j = \sum_{S \in \mathcal{S}_j} \delta_S$.
- Алгоритм поддерживает неравенство $w_j + \delta_j = \sum_{S \in \mathcal{S}_j} (w_S + \delta_S) \leq 2$.
- Покажем теперь, что всегда найдутся нужные $4 \log n$ множеств.

Доказательство (продолжение)

Доказательство

- Рассмотрим такую процедуру:
 - ▶ Повторить $4 \log n$ раз: выбрать случайно не более одного множества из \mathcal{S}_j , так что каждое множество $S \in \mathcal{S}_j$ выбирается с вероятностью $\delta_S/2$.
- Такая процедура корректна, поскольку $\delta_j/2 \leq 1$.
- Рассмотрим элемент $j' \in X$, такой что $j' \notin C$ (то есть еще не покрыт).
- Его вклад в потенциал до пересчета весов равен $n^{2w_{j'}}$.
- При каждом случайном выборе множества вероятность того, что выбранное множество не содержит j' равна $1 - \frac{\delta_{j'}}{2}$.

Доказательство (продолжение)

Доказательство

- Вероятность же того, что ни одно из случайно выбранных множеств его не содержит равна

$$\left(1 - \frac{\delta_{j'}}{2}\right)^{4 \log n} \leq n^{-2\delta_{j'}}.$$

- Таким образом, мат. ожидание вклада элемента j' в потенциал после пересчета весов равно

$$n^{-2\delta_{j'}} n^{2(w_{j'} + \delta_{j'})} = n^{2w_{j'}}.$$

- Значит, мат. ожидание значения потенциала не превосходит значения потенциала изначально.
- А тогда найдутся и соответствующие $4 \log n$ множеств. \square

Доказательство корректности

Теорема

Алгоритм ONLINE-SET-COVER корректно выдает покрытие X' размера $O(|C_{\text{opt}} \log m \log n|)$.

Доказательство

- Изначально значение потенциала не превосходит n^2 .
- Потенциал также не увеличивается.
- Значит, если $w_j \geq 1$, то $n^{2w_j} \geq 1$ и $j \in C$, то есть алгоритм выдает покрытие.
- На каждой итерации выбирается не более $4 \log n$ множеств, всего же итераций не более $|C_{\text{opt}}(\log m + 2)|$.

Что мы узнали за сегодня?

Что мы узнали за сегодня?

- Online-алгоритм — алгоритм, обрабатывающий вход по мере поступления.
- Эффективность online-алгоритма оценивается относительно алгоритма, заранее знающего последовательность запросов.
- $2H_k$ -оптимальный относительно забывчивого противника алгоритм для задачи кэширования.
- $O(\log m \log n)$ -оптимальный относительно активного offline-противника алгоритм для задачи о покрытии множествами.