Лекция 1

Жадные алгоритмы

Литература: [Шен04; КФ12]

1. Пример: задача о поиске треугольника максимальной площади, сторона которого лежит на оси Ox.
2. Жадные алгоритмы и индуктивные функции [Шен04].
   - функции максимума и суммы — индуктивные
   - индуктивное расширение на примере поиска максимума произведения
3. Онлайн-алгоритмы

1.1 Пример

Программисты относят к жадным алгоритмам алгоритмы, подчиняющиеся следующему принципу. На каждом шаге алгоритм находит локально-оптимальное решение задачи (то есть, лучшее на данный момент), и хранит в памяти только его (возможно с небольшим объёмом вспомогательных данных). Алгоритм поиска максимума в последовательности с предыдущей лекции является классическим примером жадного алгоритма: на каждом шаге алгоритм помнит только максимум считанного начального отрезка последовательности и обновляет его по мере появления новых элементов последовательности.

Начнём с примера задачи, для которой жадный алгоритм устроен чуть более сложно.

Пример 6. *На вход подаются координаты точек плоскости* $(x_0, y_0), \ldots, (x_n, y_n)$*; вход заканчивается маркером конца строки. Нужно найти треугольник максимальной площади, одна из сторон которого лежит на оси* Ox *(вершины треугольника присутствуют в последовательности).*

Подобно задачам на построение в геометрии, начнём с анализа задачи. Если треугольник уже найден, то его сторона, лежащая на оси Ox, самая длинная среди сторон на Ox по всем треугольникам. Действительно, иначе можно заменить сторону треугольника на более длинную, не поменяв высоту и увеличить тем самым площадь. Рассуждая аналогично заключаем, что вершина (x, y) треугольника, не лежащая на Ox имеет максимальное значение $|y|$ — иначе можно заменить вершину и увеличить высоту.

Проведя анализ заключаем, что для решения задачи достаточно найти самую длинную сторону на оси Ox и точку, не лежащую на оси Ox с максимальным $|y|$. Чтобы найти сторону достаточно найти самую левую и самую правую точки: то есть точки $(x_{\min}, 0)$ и $(x_{\max}, 0)$.

Для решения задачи можно воспользоваться «принципом чайника» — так программисты называют использование уже готового решения. Согласно анекдоту, чтобы программисту вскипятить чайник, ему нужно взять чайник, налить в него воду и нажать на кнопку. Но если программисту дать чайник с водой, то вместо того, чтобы нажать на кнопку, он выльет воду и сведёт тем самым задачу к предыдущей.

Итак, чтобы решить задачу, достаточно найти два максимума (среди точек вида $(x, 0)$ и вида $(x, y), y \neq 0$ по y) и один минимум — ясно, что минимум можно искать также как и максимум. Для этих целей достаточно считать все координаты в массив и выполнить три линейных алгоритма. Таким образом, мы получили линейный по времени алгоритм для решения задачи.

Этот алгоритм не считается жадным, потому что решение он находит только в самом конце и хранит много лишней информации в памяти — $\Theta(n)$ битов. Однако этот алгоритм легко преобразовать в жадный алгоритм, линейный по времени и использующий константную память.

Вместо того, чтобы искать максимум и минимум последовательно, будем искать их параллельно. Считав координаты (x, y) очередной точки, определим, лежит ли она на оси Ox, и если да, то меньше ли x текущего минимума $x_{\min}$ или больше ли x текущего максимума $x_{\max}$; если же точка не лежит на оси Ox (т. е. $y \neq 0$), то проверим больше ли $|y|$ текущего $|y|_{\max}$.

Заметим, что часть доказательства корректности была приведена выше при анализе задачи, а оставшаяся часть состоит в повторении доказательство корректности для алгоритма поиска максимума в последовательности.

Пока мы используем неформальное понятие жадного алгоритма: алгоритм жадный, если дня него выполняется принцип «дают бери, а бьют — беги». Формальное определение опирается на понятие матроида, которое сложно для нашего курса. Мы рекомендуем познакомиться с ним только после изучения нашего курса, а пока рассмотрим один из способов формализации «жадности», не претендующий на полноту.

1.2 Индуктивные функции

Рассмотрим функции, которые определены на конечных последовательностях произвольной длины $(x_1, \dots x_n)$ с элементами из множества A, и принимают значение в множестве B. Функция f данного вида называется *индуктивной*, если существует функция $F : B \times A \to B$, такая что

$$f(x_1, \dots, x_n, x_{n+1}) = F(f(x_1, \dots, x_n), x_{n+1}).$$

Пример 7. *Функции* $\max$ *и* $\mathrm{sum}(x_1, \dots, x_n) = \sum_{i=1}^{n} x_i$ *являются индуктивными:*

- $\max(x_1, \dots, x_n, x_{n+1}) = \max(\max(x_1, \dots, x_n), x_{n+1})$
- $\mathrm{sum}(x_1, \dots, x_n, x_{n+1}) = \mathrm{sum}(\mathrm{sum}(x_1, \dots, x_n), x_{n+1})$

Для каждой функции из примера $f = F$, но в общем случае это необязательно.

Жадный алгоритм можно получить для задачи, которая состоит в вычислении индуктивной функцию f, путём нахождения функции F. Если F известна, то достаточно в цикле считывать следующий элемент последовательности и вычислять $y_i = F(y_{i-1}, x_i)$, где значение $y_{i-1} = f(x_1, \dots, x_i)$ было вычислено на предыдущем шаге цикла.

Однако, часто бывает, что требуемая функция не является индуктивной, но если её чуть-чуть поправить, то она будет уже индуктивной.

Пример 8. *Функция* $f(x_1, \dots x_n) = \max\limits_{i \neq j} x_i \times x_j$*, определённая на положительный целых числах, не индуктивная. Однако её можно превратить в индуктивную, если хранить в памяти пару* (m_1, m_2) *из первого и второго максимума последовательности.*

Этот приём называют индуктивным расширением. Формально, индуктивная функция g называется *индуктивным расширением* функции f, если существует такая функция $t : B \to B$, что

$$t(g(x_1, \dots, x_n)) = f(x_1, \dots, x_n).$$

Для примера с максимальным произведением можно, взять в качестве g функцию, возвращающую пару (m_1, m_2), тогда $t(m_1, m_2) = m_1 \times m_2$.

Пример 9 (продолжение примера 6)**.** *Пусть* $f((x_1, y_1), (x_2, y_2), \dots, (x_n, y_n))$ *возвращает максимальную площадь треугольника, одна из сторон которого лежит на оси* Ox*. Построим для неё индуктивное расширение* g*.*

Функция g возвращает координаты трёх точек $((x_{\max}, 0), (x_{\min}, 0), (x, y))$, которые являются вершинами треугольника максимальной площади. Чтобы определение g было формально корректным потребуем, что бы в случае, если одна из подходящих точек не была ещё считана, в этой компоненте было записано число -1, а не пара чисел. Функция g индуктивна: построим для неё функцию G, такую что

$$G(g((x_1, y_1), (x_2, y_2), \dots, (x_{n-1}, y_{n-1})), (x_n, y_n)) = g((x_1, y_1), \dots, (x_n, y_n)).$$

Функция G получат на вход выход $((x_{\max}, 0), (x_{\min}, 0), (x, y))$ функции g и точку (x_n, y_n). Если $y_n = 0$, то в случае если $x_n > x_{\max}$, $x_{\max}$ меняется на x_n, а если $x_n < x_{\min}$, то $x_{\min}$ меняется на x_n. Если же $y_n \neq 0$, то если $|y_n| > |y|$, то точка (x, y) меняется на (x_n, y_n). В остальных случаях, выход функции g остаётся без изменений. Корректность такого пересчёта ясна из корректности исходного алгоритма; в этом примере мы просто формализовали его через индуктивные функции. Осталось определить функцию $t(g(\dots))$, которая возвращает искомую площадь $\frac{1}{2}(x_{\max} - x_{\min})\, y$ или -1, если одна из компонент входа равна -1. □

Одним из подходов решения алгоритмических задач является выбор математического инварианта, который поддерживается в ходе исполнения программы. В случае жадных алгоритмов, такой инвариант часто получается найти, сформулировав задачу в терминах индуктивных функций (возможно с расширением). Этот подход отражён в книге [Шен04], в которой индуктивные функции освящены более подробно.

1.3 Онлайн-алгоритмы

Индуктивные функции естественным образом применяются для решения задач специального вида:

Вход: последовательность $x_1, x_2, \ldots, x_n$ (n заранее не задано);
Выход: $f(x_1, x_2, \ldots, x_n)$.

Функция f является параметром и фактически определяет задачу. Тип элементов последовательности зависит от задачи. Обратим внимание, что f определена для всех конечных последовательностях, или что то же самое на любом массиве.

Решение задачи состоит в вычислении $f(x_1, x_2, \ldots, x_n)$, но помимо итогового результата требуется вычислять значение $f(x_1, x_2, \ldots, x_k)$ после обработки каждого элемента последовательности x_k на входе.

Такие задачи часто встречаются в реальной жизни. Например, при реализации электронной очереди в банке, заранее неизвестно кто из клиентов придёт по какому вопросу, и нужно распределять всех клиентов по сотрудникам по мере прихода. Другой пример, при работе агентства недвижимости, необходимо продавать квартиру первому покупателю, пожелавшему её приобрести. Агентству было бы выгоднее подождать всех покупателей за год, узнать их предпочтения и только после этого продать максимальное число квартир по лучшим (для агентства) ценам. Но покупателей приходится обслуживать в порядке их прихода.

Алгоритмы для таких задач называют *онлайн-алгоритмами*. То есть, онлайн-алгоритм вычисляет значение $f(x_1, x_2, \ldots, x_k)$ после считывания каждого элемента x_k.

Заметим, что если алгоритм на каждом шаге вычисляет индукитвную функцию, то это онлайн алгоритм. Если же он вычисляет индуктивное расширение g, то он не обязательно онлайн — для того, чтобы стать онлайн-алгоритмом, ему нужно ещё на каждом шаге вычислять и значение $t(g(x_1, \ldots, x_i))$.

Онлайн-алгоритмы возникают не только при изучении жадных задач. Например, на практике бывают нужны онлайн-алгоритмы сортировки. Такие алгоритмы на каждом шаге хранят в памяти отсортированный начальный отрезок последовательности. Представьте, что в библиотеку за неделю завозят несколько партий книг. Конечно, чтобы все их расставить на полки лучше знать заранее какие книги и сколько привезут (чтобы оставить нужное место на полках), но если этого не знать, то

книги всё равно нужно расставить на полки в отсортированном порядке, дабы обслуживать читателей. Возможно, этот пример станет более убедительным, если мы заменим библиотеку на базу данных.

Офлайн-алгоритмами называют алгоритмы, которые решают задачи обработав весь вход. Ясно, что производительность офлайн-алгоритмов не хуже, чем онлайн (первые могут работать как онлайн). Для некоторых практических задач нужны именно онлайн алгоритмы, поэтому в computer science исследуют отношение производительности онлайн-алгоритмам к офлайн для этих задач.

1.4 Связь между ключевыми понятиями лекции

У нас были формальные определения индуктивной функции (с расширением) и онлайн-алгоритмов. Мы не привели формального определения жадного алгоритма в силу его трудности для вводного курса. Заметим, что эти понятия находятся в общем положении: жадные алгоритмы используются для задач оптимизации (таких как поиск максимума какой-то функции), и эти задачи могут как иметь, так и не иметь формулировку, требуемую для онлайн-алгоритмов. Онлайн-алгоритмы, в свою очередь, могут применяться не для задач оптимизации: например, для задачи сортировки.

Индуктивные функции удобно использовать для поиска инварианта, пересчёт которого по мере обработки данных приводит к решению задачи. С их помощью легко найти жадный алгоритм для элементарных задач. В то же время, в формулировке задач для онлайн-алгоритмов фактически используется индуктивная функция.

Мы будем использовать понятие жадного алгоритма неформально. У студентов возникают сомнения: является ли алгоритм в их решении жадным? Если при решении задачи была получена индуктивная функция (возможно с расширением), то мы считаем (для простоты), что в решении получился жадный алгоритм. Хотя жадными (с точки зрения математики) бывают не только такие алгоритмы, а с точки зрения программистов, жадность — понятие растяжимое.