Тест по прошлой лекции

https://forms.gle/KdJPwNywTgjiion89

Сложность алгоритмов

Асимптотический анализ

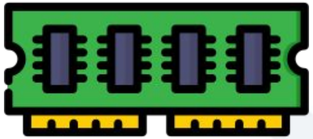
Направление «Искусственный интеллект и наука о данных», 23.Б16-мм, 23.Б18-мм

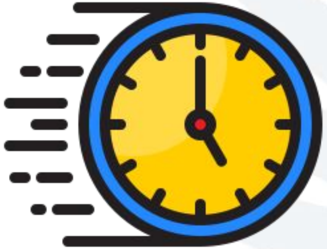
03.11.2023

Абрамов Максим Викторович, Есин Максим Сергеевич

Что будем асимптотически анализировать?

Санкт-Петербургский государственный университет





- Основными показателями **сложности** алгоритма является ***затраченное время*** работы и ***используемая память*** при работе
- Вспомним, что любой алгоритм обладает **свойством универсальности**, то есть должен подходить под несколько входных данных
- При оценке этих двух показателей важно учитывать, **насколько большие входные данные пришли на вход алгоритма.** Можно считать, что *затраченное время* и *использованная память* – это **функции от размера входных данных**

Почему анализ асимптотический?

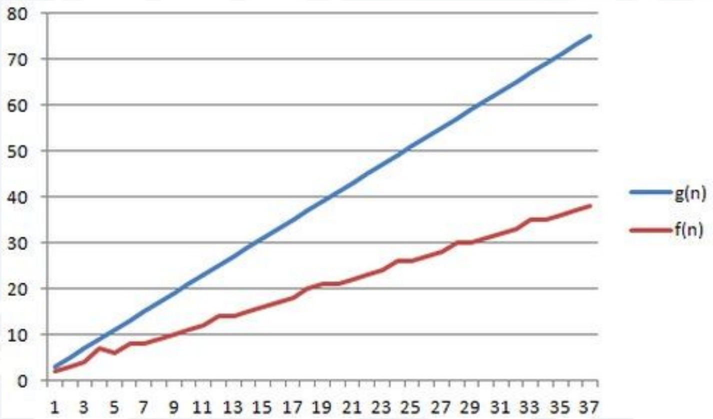
- Поскольку выяснили, что показатели сложности – функции (кстати, не обязательно одноместные) от размеров входных данных, **будем оценивать поведение функции на бесконечности**

- Для оценки поведения функции на бесконечности можно использовать **О-большое**

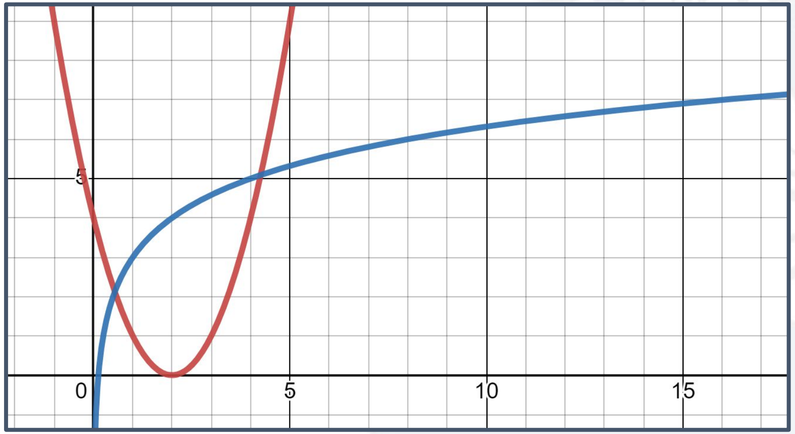
- **<u>Определение.</u>** Пусть функции f(x) и g(x) определены на R. Тогда **функцию f называют O(g)**, если:

$$\exists\, C \in R\ \forall\ x > x_0 : |f(x)| \le |Cg(x)|$$

- Функция Cg(x) выступает своего рода **асимптотой для функции f(x)**, которую она не сможет превысить



Примеры для O-большого

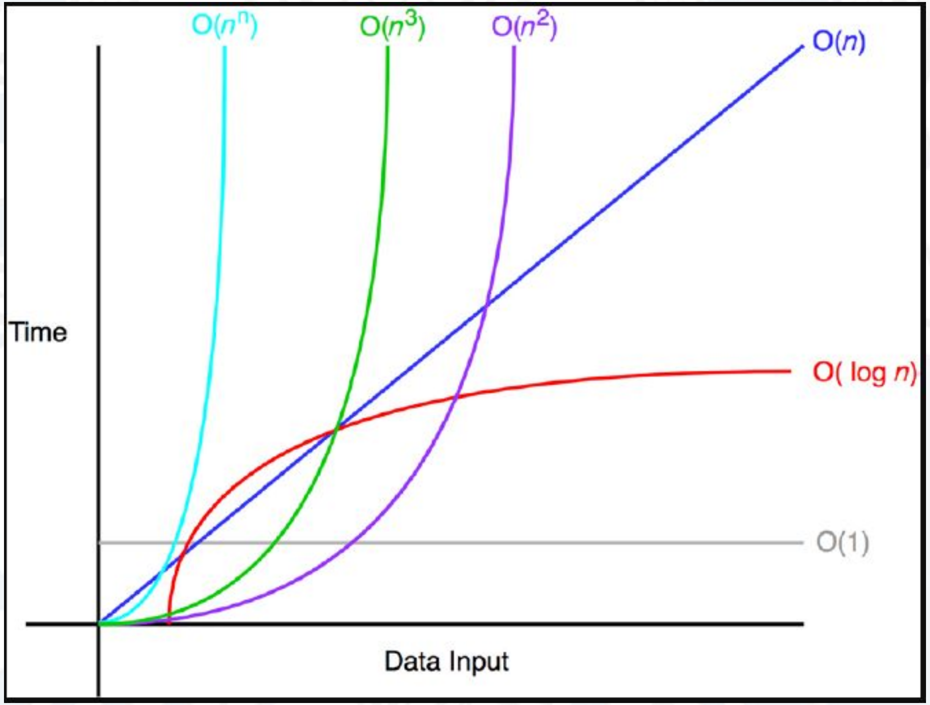


Красным отмечена $f(n) = (n - 2)^2$
Синим отмечена $g(n) = 3 + \log_2 n$

- **Самый простой пример**: $3n^2 = O(n^2)$. Можно подобрать такую константу C (например, C=4), что $Cn^2 > 3n^2$ для всех достаточно больших n
- Кроме того, верно, например, $n = O(n^2)$
- Возможно, запись $O(n^2)$ **проще воспринимать не как функцию**, **а как класс функций** f, для которых можно подобрать такую константу, что $|f(x)| \leq |Cn^2|$
- Отметим, что $f(n) = (n - 2)^2 \neq O(3 + \log n)$
- При этом, $g(n) = 3 + \log_2 n = O((n - 2)^2) = O(n^2)$

Распространенные классы функций

- Для сравнения приведена часть самых популярных классов функций для оценок сложности алгоритмов
- Одним из самых «медленных» классов является $O(n^n)$, он характеризует алгоритмы, использующие полный перебор возможных вариантов решения
- n! можно оценить как $O((n/e)^{(n+½)})$ по следствию из формулы Стирлинга



Примеры

Санкт-Петербургский государственный университет

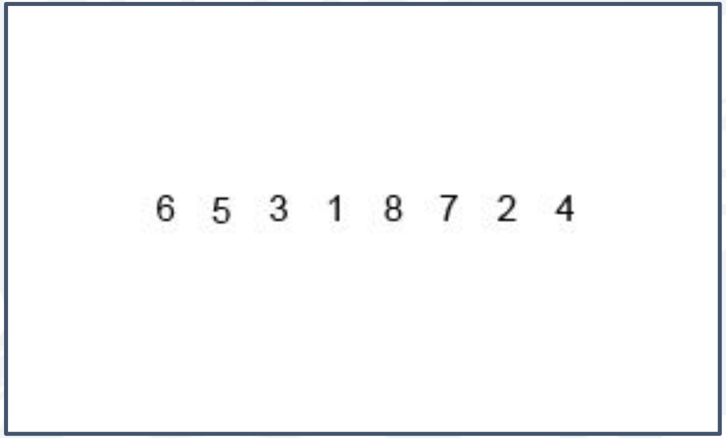
- Представим, что в памяти лежит некоторый массив, хотим **обратится к первому элементу**. **Какова сложность такой операции по времени?**
- Теперь хотим **посчитать сумму** элементов в этом массиве. Какова сложность у этого алгоритма?
- А если в массиве записаны **подряд идущие натуральные числа**, причем мы знаем их количество?
- Верно ли, что $100000n = O(n^2)$?
- Верно ли, что $O(log_3 n^2) = O(log_2 n^3)$?

Примеры

- Представим, что в памяти лежит некоторый массив, хотим **обратится к первому элементу**. **Какова сложность такой операции по времени? <u>O(1)</u>**
- Теперь хотим **посчитать сумму** элементов в этом массиве. Какова сложность у этого алгоритма? **<u>O(n)</u>**
- А если в массиве записаны **подряд идущие натуральные числа**, причем мы знаем их количество? **<u>O(1)</u>**
- Верно ли, что $100000n = O(n^2)$? **<u>Да</u>**
- Верно ли, что $O(\log_3 n^2) = O(\log_2 n^3)$? **<u>Да</u>**

Пример: сортировка слиянием

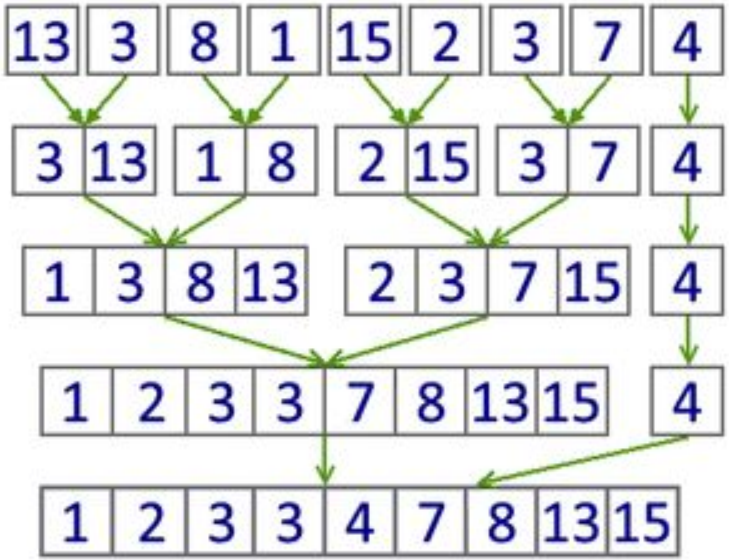
- Потренируемся оценивать сложность через О-большое на **алгоритме сортировки слиянием**
- На вход подается массив целых чисел длины n, требуется вернуть отсортированный массив
- **Алгоритм**. Разбиваем список на пары соседних чисел, сортируем числа в парах, получаем не более n/2 отсортированных пар. Далее начинаем слияние: берем две соседние пары и сливаем их в список длины 4
- На k-ой стадии алгоритма будет не более $n/2^k$ групп отсортированных чисел, которые нужно слить в $n/2^{k+1}$ групп в 2 раза большего размера



Пример работы сортировки слиянием

Пример: сортировка слиянием

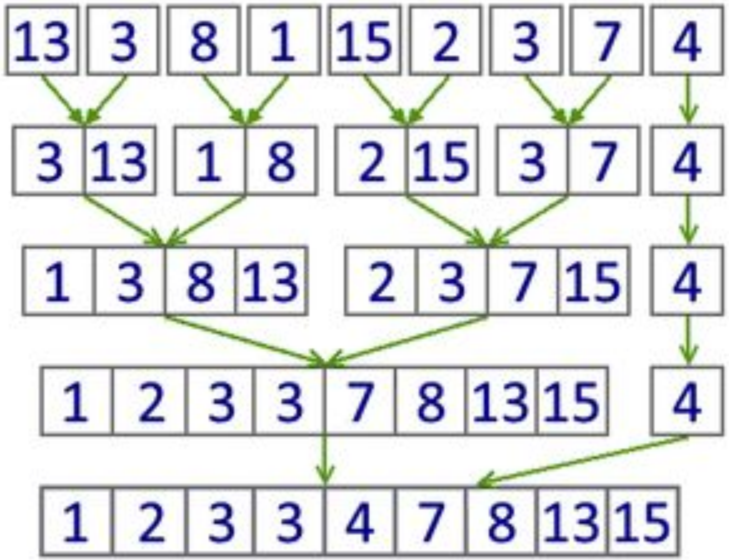
- **Оценим число стадий алгоритма** (на которых происходит слияние групп)
- Перед первой стадией число групп равно числу элементов, потому что каждая группа состоит из одного элемента
- После каждой стадии **число групп уменьшается примерно в два раза** (если быть точным, из k групп образуются *ceil(k/2)* групп. В конце остается одна группа
- Значит, **число стадий можно оценить как *O(log n)***



Пример работы сортировки слиянием

Пример: сортировка слиянием

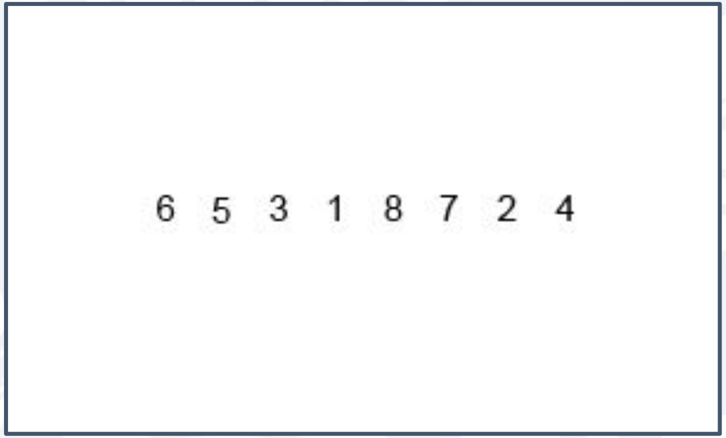
- **Что происходит на каждой из стадий**? Слияние двух групп из k элементов происходит за 4k шагов (если за шаг считать либо сравнение двух элементов, либо запись числа в итоговую группу размера 2k)
- Суммарное количество элементов в группах равно n для любой стадии, значит, **число шагов на каждой стадии можно оценить как 2n**
- Итого: на каждой из *O(log n)* стадий происходит порядка *O(n)* шагов. Значит, **временная сложность алгоритма – *O(n log n)***



Пример работы сортировки слиянием

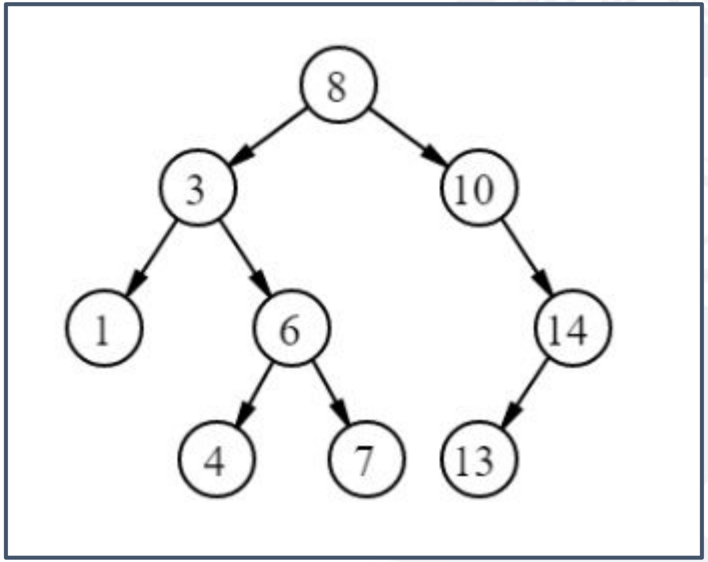
Пример: сортировка слиянием

- **Какую оценку можно дать по требуемой памяти?** Учитывая, что входной массив надо постоянно где-то хранить, то **минимальная оценка – *O(n)***
- При этом, можно показать, что **оценка сверху аналогичная**: можно хранить текущее состояние массива после очередной стадии в одном массиве, а само слияние проводить в другом
- **Сложность алгоритма по памяти – *O(n)***



Пример работы сортировки слиянием

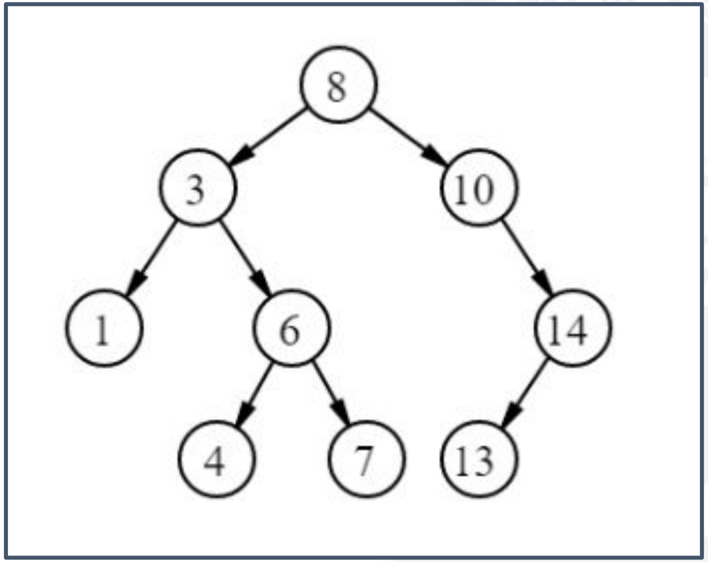
Пример: несбалансированное двоичное дерево



Двоичное дерево

- Рассмотрим популярную структуру хранения данных – **двоичное дерево**
- Суть проста: есть множество вершин, на котором задано **отношение порядка**
- У каждой вершины есть не более одного предка, а также не более двух потомков
- Особенность: для любой вершины **левый потомок должен быть меньше**, а **правый – больше**

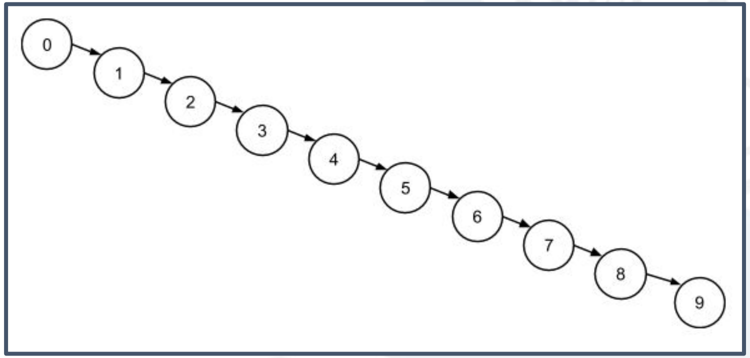
Пример: несбалансированное двоичное дерево



Двоичное дерево

- Кажется, что **операция вставки работает за *O(log n)***
- Требуется **спуститься от корня вниз**, попутно сравнивая новое значение с потомками и выбирая левое или правое поддерево, и вставить лист (*так называют вершину без потомков*)
- **Операция поиска** работает аналогично за ***O(log n)***

Пример: несбалансированное двоичное дерево



То же самое двоичное дерево

- Тогда почему мы говорим о **балансировке двоичного дерева**?
- Без балансировки после каждой операции вставки дерево может принять очень неудобный вид, который **рушит все наши предположения** о том, что операция вставки и поиска **всегда работает за *O(log n)***
- В таких случаях говорят, что алгоритм работает в ***худшем* случае за O(n)**, а **в *среднем* случае – за O(log n)**
- Балансировку дерева обеспечивает, например, АВЛ-дерево

Пример: быстрое возведение в степень

- Будем решать **задачу возведения числа 2 в положительную степень двойки** (4, 8, 16 и так далее). На вход подается единственное число N — показатель степени
- Наивный алгоритм выглядит примерно так: создаем переменную, равную единице, умножаем её на 2 нужное количество раз. **Число арифметических операций равно N**
- Модифицированный алгоритм выглядит так: ч**тобы вычислить число 2^{2k} , можно вычислить число 2^k и возвести его в квадрат**. Аналогично поступить с числом 2^k и так далее, пока не дойдем до 2^1
- Число арифметических действий модифицированного алгоритма **уменьшилось**: оно равно **$\log_2 N$, что меньше N для всех целых N > 1**

- $2^{16} = A * A$, где $A = 2^8$
- $2^8 = B * B$, где $B = 2^4$
- $2^4 = C * C$, где $C = 2^2$
- $2^2 = D * D$, где $D = 2$

Пример: быстрое возведение в степень

10 – десять
100 – сто
1 000 – тысяча
1 000 000 – миллион
1 000 000 000 – миллиард
1 000 000 000 000 – триллион
1 000 000 000 000 000 – квадриллион

- **Оценим длину полученного числа** снизу, ведь нужно куда-то его сохранить, чтобы работать с ним дальше, или просто вывести
- Заметим, что 2^{10} = 1024, что чуть больше 1000. Тогда число 2^{10*K} будет не меньше, чем 1000^K. Число 1000^K — это 3K нулей и одна единица, то есть запись числа 2^{10*K} не короче, чем 3K + 1 символ
- Тогда для входного показателя степени N **полученное число будет не короче 3N/10 + 1 символов**
- Для N = 4 это не больше 3 символов, а вот для **N = 1024 это уже минимум 307 символов**, хотя в первом случае нужно совершить 2 операции, а во втором – всего 10
- Очевидно, что процессы вывода числа длиной 3 символа и вывода числа длиной 307 символов **не могут отличаться по числу шагов всего лишь в 5 раз**

Пример: быстрое возведение в степень

- Будем хранить большие числа, разбив их на **макроцифры**, которые помещаются в один из типов данных. Каждая макроцифра отвечает сразу за несколько соседних разрядов большого числа
- Для удобства будем использовать десятичные макроцифры, не превышающие 100. Тогда **число 1 000 034 506 будет представлено в виде 6 ячеек**: 5 макроцифр и одного счетчика макроцифр
- Чтобы вывести это число, нужно вывести каждую ее макроцифру, при необходимости дополнив запись ведущими нулями. Каждая макроцифра отвечает за фиксированное число разрядов (скажем, m), поэтому для числа длины **n** нужно использовать ***O(n)*** макроцифр (не меньше n/m)
- Если считать шагом дополнение ведущими нулями и вывод макроцифры, то алгоритмическая сложность вывода большого числа равна ***O(n)*** по времени и по памяти, где **n** – д

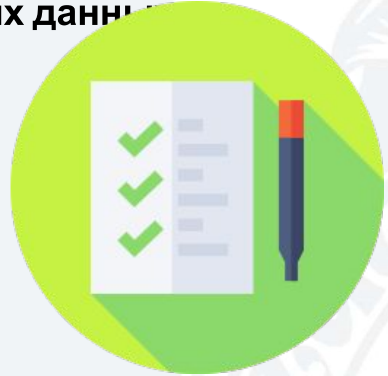
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
5	6	45	3	0	10

Пример: быстрое возведение в степень

- Следует различать вычислительную сложность алгоритма, **зависящую от длины размера входных данных**, и **«число шагов» работы алгоритма**. Это различие следует из того, что понимается под словом «шаг»
- Считая шагом любую арифметическую операцию с неограниченно большими числами, можно **неправильно оценить сложность алгоритма**
- В предыдущих алгоритмах и в дальнейшем будем считать, что **числа не очень большие**: ввод, вывод и другие арифметические операции с ними будут работать за O(1)

Итоги раздела

- Главные показатели сложности алгоритма — **затраченное время и используемая память**. Эти показатели принято оценивать как **функции от размера входных данных**
- Главный инструмент в оценке алгоритмов между собой — **О-большое**. С его помощью можно утверждать, что **одни алгоритмы будут работать лучше (быстрее) других** при стремлении размера выходных данных к бесконечности
- У алгоритмов бывают **худшие, лучшие и средние случаи работы**. Природа некоторых алгоритмов такова, что оценки по времени и памяти для этих случаев могут быть **разными**
- Следует правильно выбирать определение понятия «шаг алгоритма». Оно должно **зависеть от длины входных данн**



Сложность алгоритмов

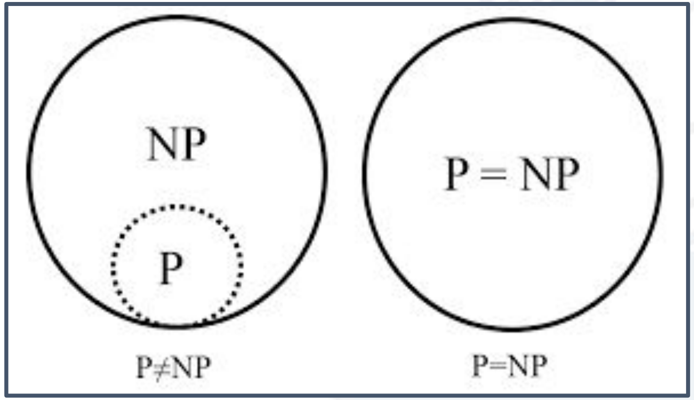
Классы P и NP

Направление «Искусственный интеллект и наука о данных», 23.Б16-мм, 23.Б18-мм

03.11.2023

Абрамов Максим Викторович, Есин Максим Сергеевич

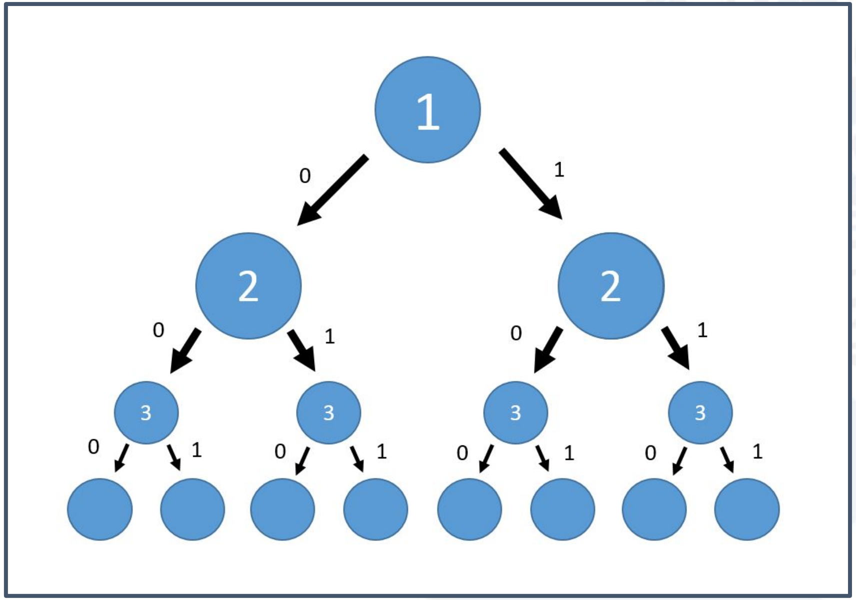
Что за классы P и NP?



Соотношение классов P и NP при равенстве и неравенстве

- Есть такие сложные задачи, для которых не придумали быстрых алгоритмов. Единственный известный способ получения *точного* решения таких задач — **проверить почти все возможные варианты**
- **Класс NP** — сложные задачи, решаемые перебором
- **Класс P** — задачи, для которых известен (квази-) полиномиальный алгоритм
- **Гипотезу о равенстве классов NP и P** пока никто не смог **ни доказать, ни опровергнуть**

Почему классы так называются?



Дерево полного перебора, которое генерирует на лентах недетерминированная МТ

- **Название класса P** происходит от английского *polynomial*, подразумевая наличие быстрого алгоритма
- **Класс NP** происходит не от non-polynomial, а от ***non-deterministic polynomial***
- Вспомним **абстрактную машину Тьюринга**: у нее была одна лента, на которой выполнялись все вычисления. Такая версия машины называется **детерминированной**
- **Недетерминированная машина** сначала генерирует все возможные варианты, выделяя каждому варианту свою ленту, а потом проверяет все ленты и ищет оптимальное решение

Как быстро работает недетерминированная машина?

- Генерация всех возможных вариантов на обычной машине Тьюринга занимает **экспоненциальное время**, например, **$O(2^n)$**
- **На недетерминированной машине**, которая размножает ленты в режиме реального времени, **генерация всех возможных вариантов занимает полиномиальное время**
- Поэтому и говорят, что задача из класса NP решается за полиномиальное время на недетерминированной машине Тьюринга

Пример задачи из класса NP

Санкт-Петербургский государственный университет

РАЗБИЕНИЕ

Дано: конечное множество A, для каждого $a \in A$ его «вес» $s(a) \in Z_+$.

Вопрос: существует ли разбиение множества A на два подмножества одинакового веса?

$$\exists A' \left(A' \subseteq A \,\&\, \sum_{a \in A'} s(a) = \sum_{a \in (A \setminus A')} s(a) \right).$$

- **Полный перебор** можно организовать следующим образом. Представим разбиение множества A как битовую полоску: 0 – если элемент входит в первое множество, 1 – если входит во второе. **Генерируем все варианты за O(2^n)**, где n – мощность входного множества
- Проверка каждого разбиения составляет O(n) операций

Итоги раздела

- Некоторые задачи **не имеют быстрых алгоритмов для решения**, например, задача о разбиении. Самый быстрый алгоритм для ***точного*** решения подобных задач – полный перебор
- Для **гипотезы о равенстве классов NP и P** пока никто не смог придумать ни доказательство, ни опровержение
- Решение задачи из класса NP **на недетерминированной МТ** занимает **полиномиальное время**, а на обычной **детерминированной — экспоненциальное**

Санкт-Петербургский государственный университет

DSCS.PRO

TICS LAB

Сложность алгоритмов

Направление «Искусственный интеллект и наука о данных», 23.Б16-мм, 23.Б18-мм

03.11.2023

Абрамов Максим Викторович, Есин Максим Сергеевич

Дополнительные лекции

Санкт-Петербургский государственный университет

- Дополнительные баллы за дополнительные лекции) От вас требуется проявить желание послушать хотя бы одну лекцию (**обязательно написав нам до конца дня**), зарегистрироваться в событии, послушать лекцию и составить конспект
- Всего можно получить от 5 до 10 баллов в зависимости от Вашей активности на лекции и качества конспекта
    - 07.11. *Показатели оценки эффективности инновационного проекта*: ссылка
    - 10.11. *Оценка бизнеса*: ссылка
    - 10.11. *Правильный питч*: ссылка
    - 23.11. *Методология «Agile»*: ссылка