6. Нижние границы эффективности алгоритмов

Разум различает возможное и невозможное; рассудок различает осмысленное и бессмысленное. Однако возможное может оказаться бессмысленным.

Макс Борн

6.1. Доказательство нижних границ

Пути оценки эффективности алгоритмов:

1. Установить класс ассимптотической эффективности и посмотреть, где он находится в иерархии классов эффективности.
2. Ответить на вопрос о том, как соотносится эффективность конкретного алгоритма с другими алгоритмами для решения той же задачи.

6.1.1 Тривиальные нижние границы

Простейший метод получения нижних границ (НГ) основан на подсчете количества элементов входных данных, которые следует обработать.

Граница называется плотной, если уже существуют алгоритм, удовлетворяющий этой нижней границе.

Пример: вычисление полинома

$P(x)=A_n x_n + A_{n-1}x_{n-1}+\ldots+A_0$ для заданных коэффициентов.

Размер входных данных: n.

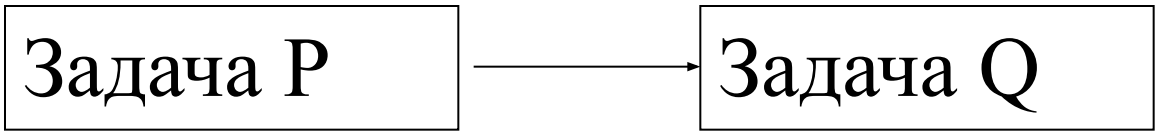
Любой алгоритм вычисления полинома должен иметь эффективность $\Omega(n)$.

Эта граница является плотной, т.к. существует схема Горнера с линейным временем работы.

6.1.2 Информационно-теоретические доказательства

Информационно-теоретический подход пытается установить нижнюю границу эффективности алгоритма на основе количества информации, которую он производит.

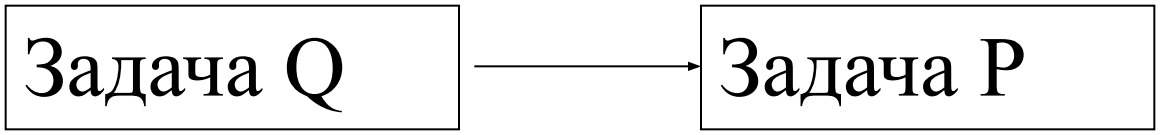
Приведение задачи



Алгоритм неизвестен

Алгоритм известен

Для поиска нижней границы:



Нижняя граница известна

Нижняя граница неизвестна

Задача	**Нижняя граница**	**Плотность**
Сортировка	$\Omega(n \log n)$	Да
Поиск в отсортированном массиве	$\Omega(\log n)$	Да
Задача единственности элемента	$\Omega(n \log n)$	Да
Умножение n-значных целых чисел	$\Omega(n)$	Неизвестно
Умножение матриц	$\Omega(n^2)$	Неизвестно

Поиск евклидова минимального остовного дерева

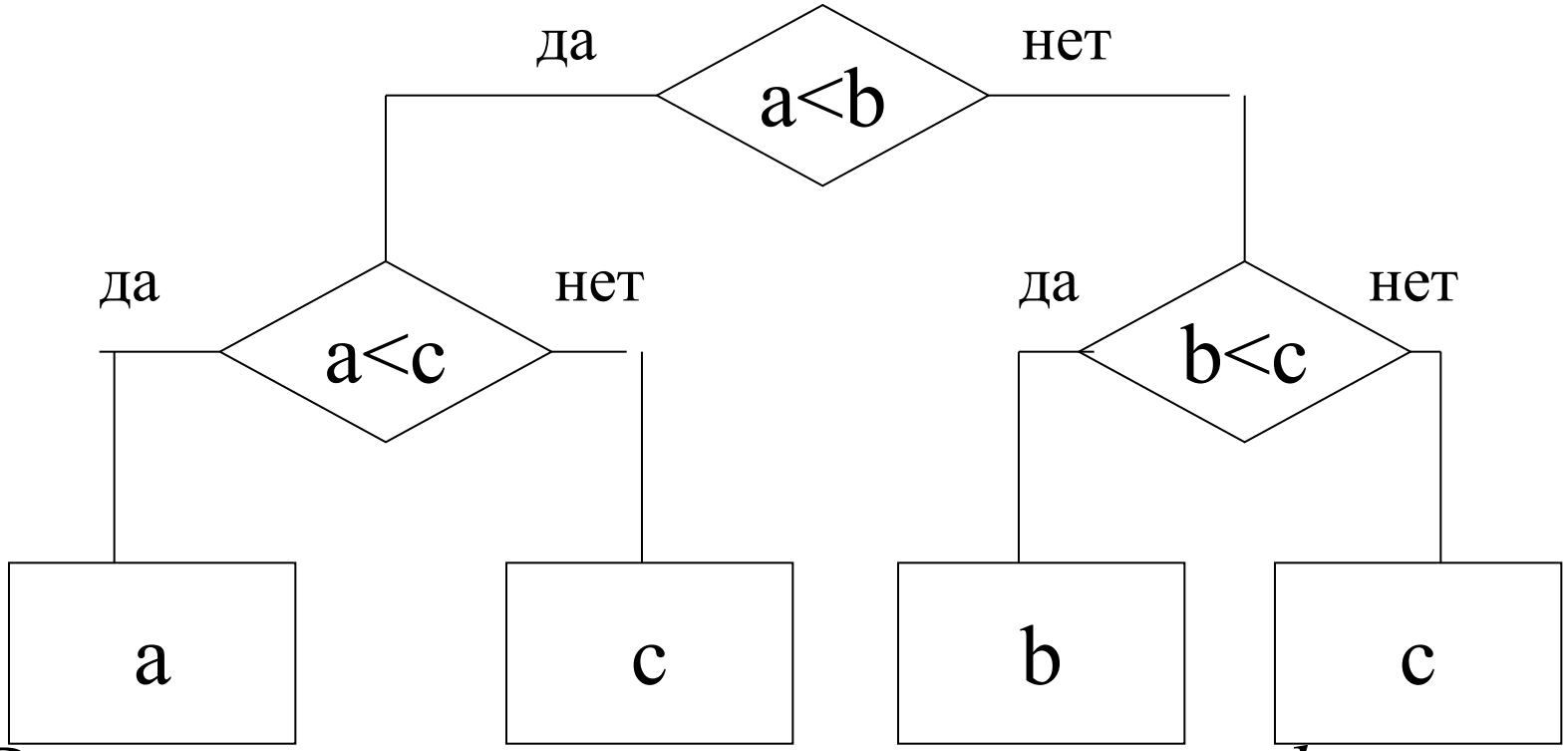
Требуется: построить дерево минимальной длины, узлами которого являются n точек на декартовой плоскости.

Задача с известной нижней границей: задача единственности элементов.

Приведение: множество из n действительных чисел $x_1, x_2 \ldots, x_n$ прелобразуется в множество точек на плоскости $(x_1,0), (x_2,0),\ldots, (x_n,0)$.

Нижняя граница: $\Omega(n \log n)$

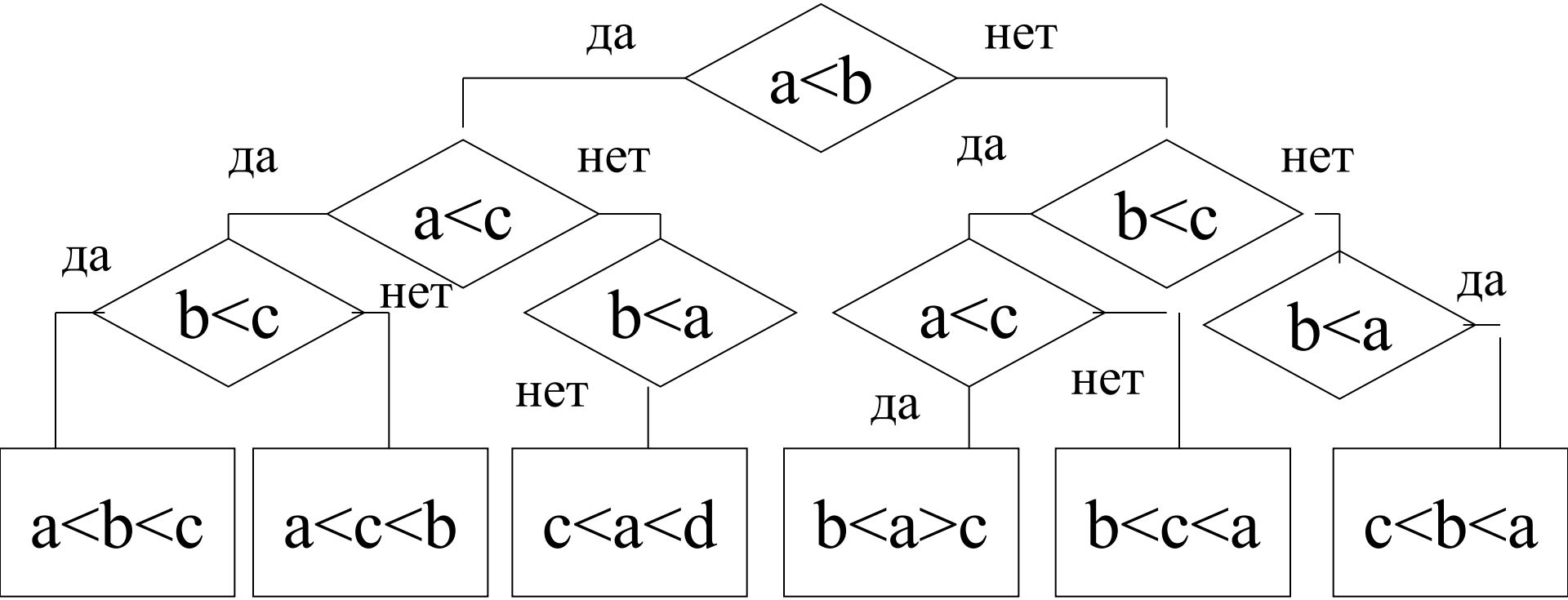
Деревья принятия решения



Высота дерева принятия решения с l листьями: $h >= \lceil \log_2 l \rceil$ (1)

Наибольшее количество листьев в таком дереве равно 2^h .

Деревья принятия решения для алгоритма сортировки



Высота дерева принятия решения для произвольного алгоритма сортировки:

$$C_{worst} >= \lceil \log_2 n! \rceil .$$

Используя формулу Стирлинга для n! получим:

$$\lceil \log_2 n! \rceil \approx \log_2 \sqrt{2\pi n}\,(n/e)^n =$$

$$= n \log_2 n - n \log_2 e + (\log_2 n)/2 + (\log_2 \sqrt{2\pi})/2 \approx$$

$$\approx n \log_2 n .$$

Вывод: необходимо примерно $n \log_2 n$ сравнений для сортировки **n**–элементного списка **любым** алгоритмом сортировки, который основан на сравнениях.

6.2. P, NP и NP-полные задачи

Определение 1. Алгоритм решает задачу за **полиномиальное** время, если его временная эффективность **в наихудшем случае принадлежит классу O(p(n))**,

где p(n) – **полином** от размера задачи n.

Задача, решаемая за полиномиальное время называется легкой, в противном случае- трудной.

n	**$\log_2 n$**	**n**	**$n\log_2 n$**	**n^2**	**n^3**	**2^n**	**n!**
10	3,3	10	3,3 10	10^2	10^3	10^3	3,6 10^6
10^2	6,6	10^2	6,6 10^2	10^4	10^6	1,3 10^{30}	9,3 10^{157}
10^3	10	10^3	10^4	10^6	10^9		
10^4	13	10^4	1,3 10^5	10^8	10^{12}		
10^5	17	10^5	1,7 10^6	10^{10}	10^{15}		

6.2.1. P и NP-задачи

Большинство ранее рассмотренных задач **могли быть решены** с применением некоторого алгоритма за **полиномиальное** время.

Эти задачи можно рассматривать как ***множество***, которое называют ***классом P***.

Более ***формальное*** определение включает в P ***только задачи принятия решения***, представляющие собой задачи, выходом которых могут быть только ответы : «да» и «нет».

Определении 2.

Класс P представляет собой класс задач принятия решения, которые могут быть решены (детерминистическим алгоритмом) за полиномиальное время.

Этот класс задач называется ***полиномиальным***.

Ограничение класса P только задачами принятия решения можно объяснить следующими причинами.

Во-первых, разумно сразу же исключить задачи, не разрешимые за полиномиальное время из-за экспоненциально большого размера входных данных. Например, задачу генерации всех подмножеств данного множества или всех перестановок n различных элементов и т.п.

Во-вторых, многие важные задачи не являются задачами принятия решения в своей естественной постановке, но могут быть ***приведены*** к ряду ***проблем принятия решения***, которые проще изучать.

Для решения некоторых задач принятия решения невозможно применение вообще никаких алгоритмов. Такие задачи называются ***алгоритмически неразрешимыми***.

Задача останова (Алан Тьюринг, 1936 г.) **Суть** её состоит в следующем:

для данной компьютерной программы необходимо определить, завершится ли выполнение программы или она будет выполняться бесконечно.

Доказательство неразрешимости этой проблемы от противного.

Предположим, что A – алгоритм, который **решает** задачу останова, то есть для любой программы P и входных данных I.

Функция преобразования входных данных в выходные может быть описана следующим образом:

$$A(P,I) = \begin{cases} 1, & \text{если программа } P \text{ завершается при входных данных } I \\ 0, & \text{если программа } P \text{ не завершается при входных данных } I. \end{cases}$$

Будем рассматривать программу P как **входные данные** для неё самой и использовать **выход алгоритма A** для пары (P, P) для построения программы Q следующим образом:

Q(P)= Завершается, если A(P,P)=0, т.е. программа P не завершается при входных данных P;

Не завершается, если A(P,P)=1, т.е. программа P завершается при входных данных P.

Подставляя вместо P программу Q, получим:

Q(Q)= { Завершается, если A(Q,Q)=0, т.е. программа Q не завершается при входных данных Q;
Не завершается, если A(Q,Q)=1, т.е. программа Q завершается при входных данных Q.

Мы пришли к **противоречию**, поскольку ни один из выходов программы Q невозможен, что и завершает доказательство.

Важные задачи, для которых **не найден алгоритм с полиномиальным временем работы**, но **не доказана невозможность его существования**:

- ***Задача построения гамильтонова цикла***;
- ***Задача о коммивояжере.*** Требуется найти кратчайший маршрут по n городам с известными целочисленными расстояниями между ними;
- ***Задача о рюкзаке.*** Обсуждалась в разделе 5;
- ***Задача о разделении.*** Даны n положительных целых числа. Требуется определить, можно ли разделить их на два непересекающихся подмножества с одинаковыми суммами;
- ***Задача об упаковке корзин.*** Даны n предметов, размеры которых представляют собой положительные рациональные числа, не превышающие единицу. Их необходимо разместить в наименьшем количестве корзин ёмкостью единица;
- ***Задача о раскраске графа.*** Обсуждалась в разделе 5;
- ***Задачи целочисленного линейного программирования.*** Требуется найти максимальное (минимальное) значение линейной функции нескольких целочисленных переменных при условии выполнения конечного множества ограничений в виде линейных равенств и (или) неравенств.

Определение 3.

Недетерминистическим алгоритмом (НДА) называется двухэтапная процедура, которая получает в качестве входа экземпляр I задачи принятия решения и делает следующее:

- ***недетерминистический*** этап («угадывание»): генерируется строка S, которая рассматривается в качестве кандидата на решение I;
- ***детерминистический*** этап («проверка»): детерминистический алгоритм получает I и S в качестве входных данных и выдает «да», если S – решение задачи I и «нет» – в противном случае.

НДА является ***недетерминистическим полиномиальным*** (НДП), если временная эффективность этапа проверки полиномиальная.

Определение 4.

Класс NP – это класс задач принятия решения, которые могут быть решены НДП алгоритмом.

Этот класс задач называется ***недетерминистическим полиномиальным***. Большинство задач принятия решения принадлежит классу NP.

Прежде всего этот класс включает все задачи класса P: $P \subseteq NP$.

Это соотношение истинно, так как если задача принадлежит классу P, мы можем использовать ДПА, который решает её на этапе проверки НДА, просто проигнорировав строку S, генерируемую на этапе «угадывания».

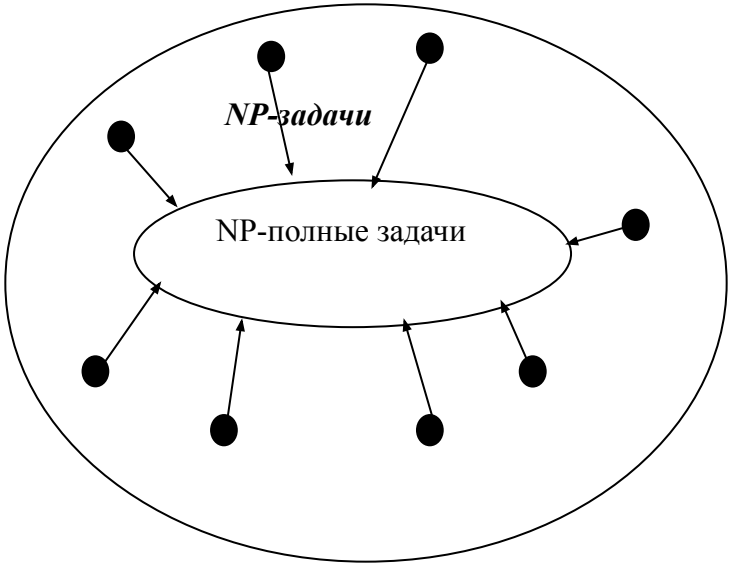
Класс NP включает также такие задачи, как задача поиска гамильтонова цикла и т.п. С другой стороны, **задача останова** находится среди тех задач принятия решения, о которых известно, что они **не входят в класс NP**.

Это приводит к наиболее важному открытому вопросу теоретической информатики: является ли ***класс P истинным подмножеством NP*** или на самом деле ***оба этих класса совпадают, т.е. P≡NP.***

Истинность утверждения ***P≡NP*** должна приводить к тому, что **каждая** из многих сотен **задач принятия решения** может быть решена с использованием алгоритма с **полиномиальным временем работы**, хотя для многих подобных задач такие алгоритмы **не найдены** несмотря на многолетние усилия.

Кроме того, многие хорошо известные задачи принятия решения являются ***NP-полными***.

Приведение NP-задач к NP-полным задачам



6.2.2 NP-полные задачи

Определение 5.

Задача принятия решения $З_1$ называется ***полиномиально приводимой*** к задаче принятия решения $З_2$, если имеется функция t, которая преобразует экземпляры $З_1$ в экземпляры $З_2$ так, что

- t отображает все экземпляры $З_1$ с положительным ответом на экземпляры $З_2$ с положительным ответом, и все экземпляры $З_1$ с отрицательным ответом на экземпляры $З_2$ с отрицательным ответом;
- t вычислима при помощи алгоритма с полиномиальным временем работы.

Определение 6.

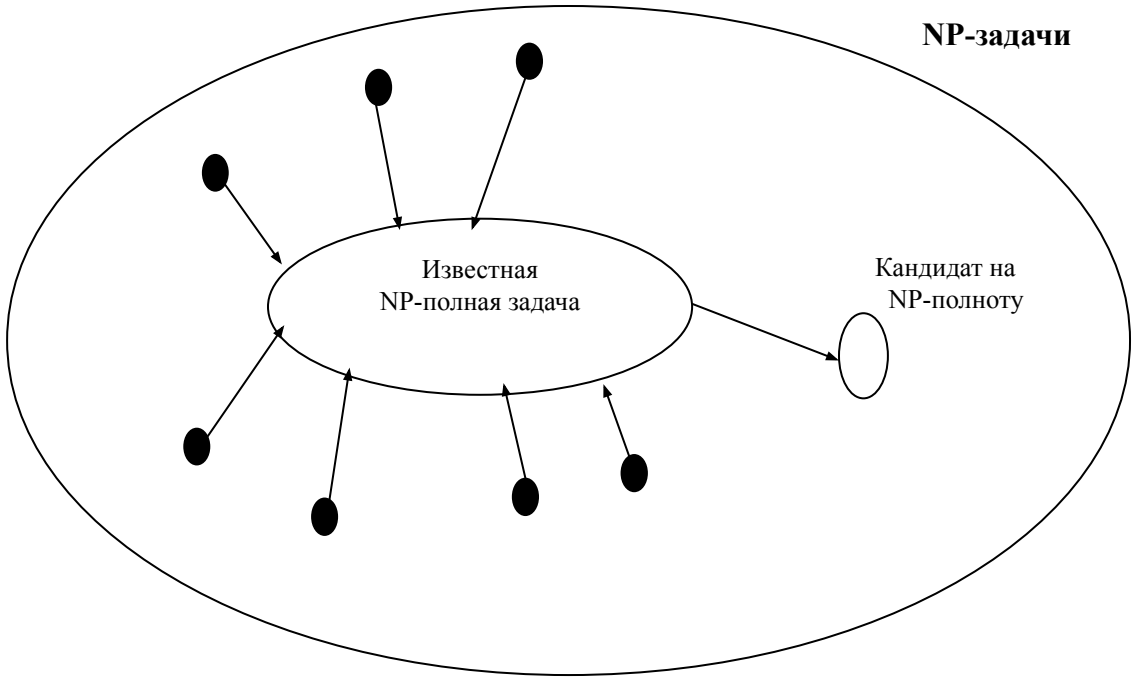
Задача принятия решения D является ***NP-полной***, если

- она принадлежит классу NP и
- любая задача NP полиномиально приводима к D.

  Понятие NP-полноты требует полиномиальной приводимости ***всех*** задач в NP, как известных, так и неизвестных, к рассматриваемой задаче

Показать, что задача является NP-полной, можно в два этапа:

- Показать, что ***рассматриваемая задача*** является ***NP-задачей***, т.е. случайным образом сгенерированная строка может быть проверена на пригодность в качестве решения задачи за полиномиальное время;
- Показать, что любая задача из множества NP приводима к рассматриваемой задаче за полиномиальное время. Для выполнения этого этапа достаточно показать, что ***известная*** NP-полная задача может быть приведена к ***рассматриваемой*** задаче за ***полиномиальное время***.



Доказательство NP-полноты приведением.

6.3. Выводы

- Непосредственно из определения NP-полноты следует, что если будет найден детерминистический алгоритм решения ***одной*** из NP-полных задач, то ***все задачи в NP*** могут быть решены за полиномиальное время при помощи детерминистического алгоритма, следовательно, P≡NP.
- Нахождение алгоритма с полиномиальным временем работы для одной NP-полной задачи будет означать, что ***не существует*** качественного различия между ***сложностью проверки*** предлагаемого решения и ***поиска*** его за полиномиальное время для подавляющего большинства задач принятия решения всех видов.

- Каким бы ни был окончательный ответ на вопрос P=?NP, на сегодняшний день знание того, что задача является NP-полной, имеет важные практические следствия. Это означает, что, столкнувшись с NP-полной задачей, ***не стоит*** тратить массу усилий для ***разработки полиномиального алгоритма для всех её экземпляров***. Вместо этого следует сконцентрироваться ***на поиске уменьшения сложности*** поставленной задачи.