Дискретные задачи принятия решений

НГУ • Механико – математический факультет • 4 курс

Лектор: **Кочетов Юрий Андреевич**

http://www.math.nsc.ru/LBRT/k5/tpr.html

Лекция 1.

Задачи комбинаторной оптимизации.

Алгоритмы и сложность

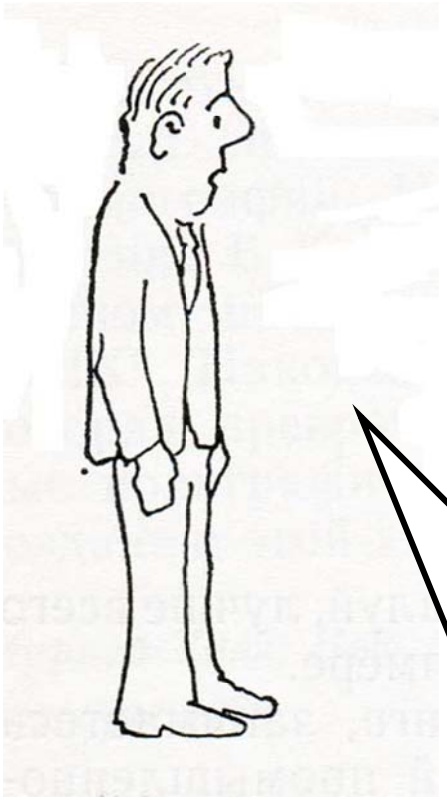
Мотивационный пример

Компания планирует производство и продажу бандерснечей в условиях жесткой конкуренции.

Нужен метод для проверки возможности создания узлов бандерснеча с заданными техническими характеристиками.

Вы отвечаете за разработку алгоритмов.

Мрачная перспектива



Я не могу найти эффективного решения. Боюсь, что я для этого слишком туп.

Удобный выход

Но доказательство может оказаться слишком сложной задачей, если это вообще можно доказать.

Теория NP-полных задач дает методы доказательства того, что конкретная задача столь же трудна, как и ряд других задач, признанных очень трудными

Достойный выход

- Проверить NP–трудность (полноту) и постараться найти.
- Эффективные алгоритмы для частных случаев.
- Эффективные приближенные алгоритмы.
- Алгоритмы, работающие эффективно в среднем.
- Итерационные методы (метаэвристики), позволяющие в асимптотике находить точное решение задачи.
- Другие идеи, например, искать быстрые алгоритмы, гарантирующие выполнение бóльшей части ограничений задачи.

Массовая задача

Под *массовой задачей* (или просто *задачей*) будем понимать некоторый общий вопрос, на который следует дать ответ:

– Есть ли в задаче гамильтонов цикл?

– Существует ли в графе клика мощности не менее K?

Задача задается следующей информацией:

1) списком всех ее параметров;

2) формулировкой свойств, которым должен удовлетворять ответ

Индивидуальная задача получается из массовой присвоением конкретных значений всем параметрам.

Алгоритмы

Под *алгоритмом* будем понимать общую, выполняемую шаг за шагом процедуру решения задачи. Для определенности можно считать ее программой на Си или другом языке.

Будем выделять

- точные алгоритмы, которые для любой индивидуальной задачи всегда дают точное решение;
- приближенные алгоритмы с гарантированной оценкой точности;
- аппроксимационные схемы — семейство алгоритмов, позволяющих получать решения с любой наперед заданной точностью $\varepsilon > 0$, время работы которых растет с ростом величины $1/\varepsilon$;
- итерационные методы локального поиска (метаэвритстики), для которых вероятность получить точное решение растет с ростом числа итераций;
- быстрые конструктивные эвристики без гарантии получить точное решение или решение с заданной погрешностью.

Длина входа индивидуальной задачи

Время работы алгоритма и требующуюся память выражают в виде функции от «размера» индивидуальной задачи, т. е. объема входных данных.

Входная длина (размер) индивидуальной задачи есть число символов, необходимых для кодирования ее исходных данных.

Пример.

Для задачи коммивояжера требуется задать $n \times n$ матрицу расстояний (c_{ij}). При двоичном кодировании чисел длина входа (размер) не превышает величины $n^2 \max_{ij} (\log_2 c_{ij})$.

Полиномиальные алгоритмы

Будем говорить, что функция $f(n)$ есть $O(g(n))$, если существует такая константа c, что $|f(n)| \leq c|g(n)|$ для всех $n \geq 0$.

Полиномиальным алгоритмом (алгоритмом полиномиальной временной сложности) называется алгоритм, у которого временная сложность равна $O(p(n))$, где $p(n)$ — некоторая полиномиальная функция, а n — длина входа.

Алгоритмы, временная сложность которых не поддается подобной оценке, называются *экспоненциальными*.

Пример. $O(n^2)$ — полиномиальная сложность;

$O(n^{\log n})$ — экспоненциальная сложность.

Сравнение полиномиальных и экспоненциальных функций временной сложности

Функция временной сложности	Размер n					
	10	20	30	40	50	60
n	0,00001 сек.	0,00002 сек.	0,00003 сек.	0,00004 сек.	0,00005 сек.	0,00006 сек.
n^2	0,0001 сек.	0,0004 сек.	0,0009 сек.	0,0016 сек.	0,0025 сек.	0,0036 сек.
n^3	0,001 сек.	0,008 сек.	0,027 сек.	0,064 сек.	0,125 сек.	0,216 сек.
n^5	0,1 сек.	3,2 сек.	24,3 сек.	1,7 мин.	5,2 мин.	13 мин.
2^n	0,001 сек.	1,0 сек.	17,9 мин.	12,7 дней	35,7 лет	366 столетий
3^n	0,059 сек.	58 мин.	6,5 лет.	3855 столетий	2×10^8 столетий	$1{,}3\times10^{13}$ столетий

Влияние технического прогресса

Функция временной сложности	На современных PC	На ЭВМ, в 100 раз более быстрых	На ЭВМ, в 1000 раз более быстрых
n	N_1	$100\ N_1$	$1000\ N_1$
n^2	N_2	$10\ N_2$	$31{,}6\ N_2$
n^3	N_3	$4{,}64\ N_3$	$10\ N_3$
n^5	N_4	$2{,}5\ N_4$	$3{,}98\ N_4$
2^n	N_5	$N_5 + 6{,}66$	$N_5 + 9{,}97$
3^n	N_6	$N_6 + 4{,}19$	$N_6 + 6{,}29$

Быстрая сортировка

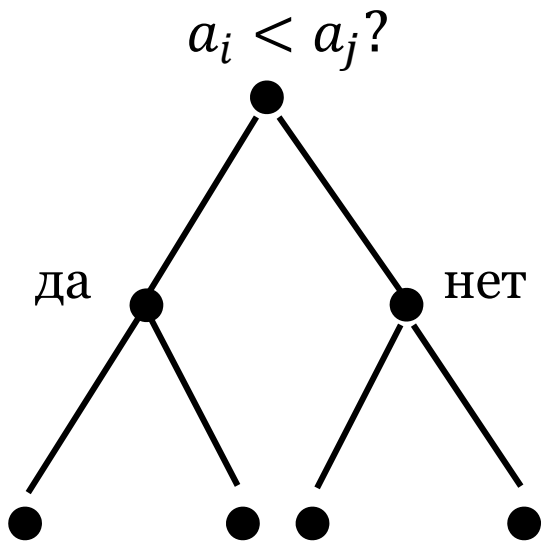
Сортировкой называют упорядочение множества объектов по неубыванию или невозрастанию какого-нибудь параметра.

- Алгоритм пузырька (волновой алгоритм) — $O(n_2)$.
- Алгоритм Фон–Неймана — $O(n \log n)$.
- Пирамидальный алгоритм — $O(n \log n)$.
- QuickSort и др.

Сортировка с помощью сравнений

Лемма 1. Бинарное дерево высоты h содержит не более 2^h листьев.

Дерево решений:



Лемма 2. Высота любого дерева решений, упорядочивающего последовательность из *n* различных элементов, не менее $\log n!$.

Доказательство. Так как результатом может быть любая из $n!$ перестановок, то в дереве решений должно быть не менее $n!$ листьев. Тогда по лемме 1 высота дерева не меньше $\log n!$. ■

Теорема. В любом алгоритме, упорядочивающем с помощью сравнений, на упорядочивание последовательность из n элементов тратится не менее $c\,n\,\log n$ сравнений при некотором $c > 0$ и достаточно большом n.

Доказательство. При $n \geq 4$ имеем

$$n! \geq n(n-1)(n-2)\ldots\left(\left\lceil \frac{n}{2} \right\rceil\right) \geq \left(\frac{n}{2}\right)^{n/2},$$

тогда

$$\log n! \geq \left(\frac{n}{2}\right)\log\left(\frac{n}{2}\right) \geq \left(\frac{n}{4}\right)\log n.$$

■

Алгоритм Фон–Неймана

На вход подается последовательность чисел $a(1), \dots, a(n)$. Алгоритм работает $\lceil \log_2 n \rceil$ итераций. Перед началом итерации с номером k $(k = 1, 2, \dots, \lceil \log_2 n \rceil)$ имеется последовательность $a(i\,(1)), \dots, a(i\,(n))$ тех же чисел, разбитая на группы по 2^{k-1} элементов (последняя группа может быть неполной). Внутри каждой группы элементы упорядочены по неубыванию. Итерация состоит в том, что эти группы разбиваются на пары соседних групп, и элементы упорядочиваются внутри этих новых в два раза больших групп. При этом используется то, что внутри исходных групп элементы уже упорядочены.

$$\left.\begin{matrix} x_1, \dots, x_m \\ y_1, \dots, y_m \end{matrix}\right\} \Longrightarrow z_1, \dots, z_{2m}$$

слияние за линейное время $O(m)$

Упражнение. Показать, что алгоритм Фон–Неймана использует $O(n\lceil \log n \rceil)$ сравнений.

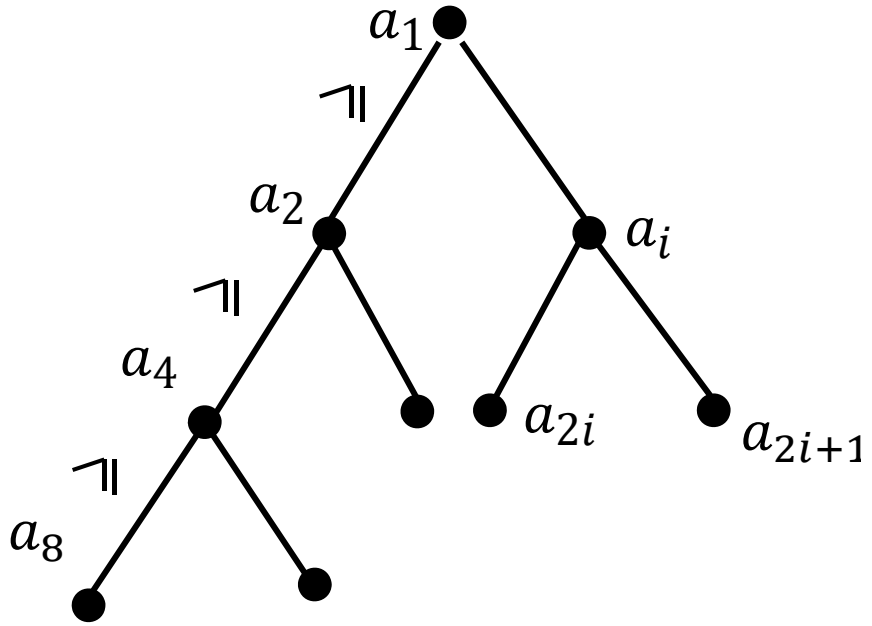
Пирамидальный алгоритм

Два этапа:

1) построение пирамиды: для каждого i

   $a_i \leq a_{2i}$ и $a_i \leq a_{2i+1}$

   сортировка массива с помощью пирамиды



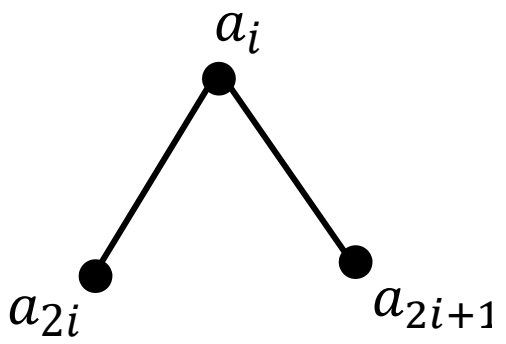
Первый этап

(i, n)–операция состоит в следующем:

Сравниваем a_{2i} и a_{2i+1},

пусть x — меньшее из них.

Если $a_i > x$, то меняем местами a_i и x.



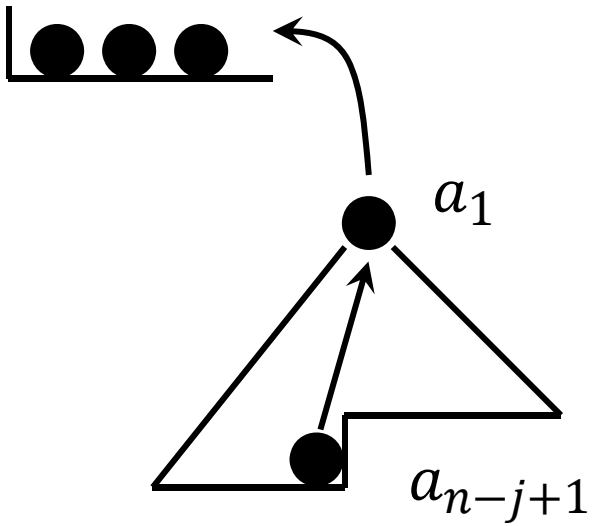
(j, n)–процедура: выполняем (j, n)–операцию; если a_j переместилось вниз, скажем, стало a_{2j+1}, то производим $(2j + 1, n)$–операцию и т.д., пока наш элемент a_j не остановится.

Первый этап состоит в последовательном выполнении (j, n)–процедуры для $j = \lceil n/2 \rceil, \lceil n/2 \rceil - 1, \ldots, 1$.

Упражнение. Доказать, что первый этап требует не более $2n(i, n)$–операций.

Второй этап

На j-й итерации $(j-1)$ самых малых чисел уже найдены и лежат на «полочке» в нужном порядке. Остальные находятся в пирамиде $(a_i \le \min(a_{2i}, a_{2i+1}))$. Итерация состоит в том, что элемент a_1 из пирамиды кладется на «полку», а на его место ставится элемент a_{n-j+1} из пирамиды и выполняется $(1, n-j)$–процедура.



После выполнения n-й итерации все числа лежат на полке в полном порядке.

Время — $O(n \log n)$.

Замечание. «Полку» можно организовать прямо в пирамиде.