3.8 Характеристики сложности алгоритмов

- **Различные подходы к уточнению понятия «алгоритм» позволяли изучать принципиальную возможность решения некоторой математической задачи.**
- **Однако теоретическая возможность алгоритмического решения задачи еще не гарантирует практическую реализуемость алгоритма.**

Характеристики алгоритмов

- показывают степень практической реализуемости алгоритмов
- позволяют сравнивать эффективность различных алгоритмов, решающих одну и ту же задачу

- При решении некоторой задачи *P* алгоритмом *A* обычно рассматривают такие характеристики

- количество шагов $T_A(x)$, которое необходимо сделать алгоритму для получения результата при использовании входных данных x.

- **Величина** $$T(A,n) = \max_{|x|=n} T_A(x)$$

(максимум берется по всем входным данным объема *n*) называется *временной сложностью* алгоритма *A*.

- объем памяти $M_A(x)$, необходимый для хранения всех входных и промежуточных данных в процессе выполнения алгоритма при использовании входных данных x.

- **Величина** $M(A,n)=\max\limits_{|x|=n} M_A(x)$

(максимум берется по всем входным данным объема *n*) называется *емкостной сложностью* алгоритма *A*.

- Аналогично можно определить *средние* величины временной и емкостной сложности алгоритма.

- **Сложность задачи P – это сложность наилучшего алгоритма, известного для ее решения, т.е.**

$$S(P,n) = \min_{A} T(A,n)$$

(минимум берется по всем алгоритмам для задачи).

- **Для определения временной сложности алгоритма вместо общего числа шагов алгоритма можно также использовать количество операций определенного вида.**

- **Арифметическая функция *f(x)* называется функцией *одного верхнего порядка* с функцией *g(x)*, т.е.**

$$f(x)=O(g(x)),$$

если существует такая натуральная константа C и некоторое натуральное N_0, что $|f(x)| \le C|g(x)|$ для всех $x \ge N_0$.

- **Арифметическая функция *f(x)* называется функцией *одного нижнего порядка* с функцией *g(x)*, т.е.**

$$f(x)=\Omega(g(x))\ ,$$

если существует такая натуральная константа C и некоторое натуральное N_0, что $|f(x)|\geq C|g(x)|$ для всех $x\geq N_0$.

- **Арифметическая функция *f(x)* называется функцией *одного порядка* с функцией *g(x)*, если она одного верхнего и одного нижнего порядка с функцией *g(x), т.е.***

$$f(x)=O(g(x)) \text{ и } f(x)=\Omega(g(x))$$

Пример.

- Пусть $f(x)$=log x и $g(x)$=x.
- Тогда существует положительная константа C, что log x ≤C·x при x≥1.
- Таким образом, log x = O(x)

- **Функция одного верхнего порядка с полиномиальными функциями называется *полиномиальной функцией*.**
- ***Экспоненциальными* называются функции одного нижнего порядка с экспонентой.**
- **Функции между экспоненциальными и полиномиальными называются субэкспоненциальными функциями.**

- **Арифметические функции f(x) и g(x) и называются**

  полиномиально эквивалентными,
  если существуют такие многочлены $P_1(x)$ и $P_2(x)$

  и некоторое натуральное N_0, что

  $$f(x) \le P_1(g(x)) \text{ и } g(x) \le P_2(f(x))$$

  для всех $x \ge N_0$.

Пример

- Функции $f(x)=2x+3$ и $g(x)=x^3$ полиномиально эквивалентны, поскольку
  существуют полиномы
  $P_1(x)=x$ и $P_2(x)=x^3$, что
  $f(x)\le P_1(g(x))=x^3$ и
  $g(x) \le P_2(f(x))=(2x+3)^3$

- **Рассмотрим задачу сортировки массива из элементов. Хорошо известны алгоритмы решения этой задачи.**

- **В качестве временной сложности алгоритма сортировки используют две характеристики – количество сравнений элементов T_1 и количество пересылок элементов T_2, необходимых в ходе работы алгоритма.**

- **Для метода пузырьковой сортировки показано, что**

  $T_1(n)=O(n^2)$ **и** $T_2(n)=O(n^2)$ **при** $n \to \infty$.

- Однако существуют алгоритмы, например алгоритм Хоара,

  который имеет лучшую верхнюю оценку временной сложности, чем метод пузырьковой сортировки.

В частности, *при* $n \to \infty$

$$T_1(n)=O(n\log n) \text{ и } T_2(n)=O(n\log n)$$

- **Для задачи сортировки доказана нижняя оценка временной сложности $T_1(n) \geq C_1 n \log n$ и $T_2(n) \geq C_2 n \log n$ при $n \to \infty$**
- **Таким образом, сложность задачи сортировки массива из элементов имеет порядок $n \log n$ при $n \to \infty$**

Рассмотрим известный алгоритм сложения двух чисел столбиком

- Входные данные – два числа, записанные в десятичной системе.
- Будем считать, что числа имеют *n* десятичных цифр в своем представлении.

- В качестве временной сложности этого алгоритма будем использовать количество операций сложений цифр, которое требуется для получения результата.
- Емкостная сложность – количество десятичных цифр, необходимых для сложения.

- Максимальное число сложений получается в случае, когда происходит перенос разряда для каждой цифры, т.е. $T(n)=n+n+1=2n+1$
- Емкостная сложность $M(n)=O(n)$.

- **Определим сложность задачи сложения двух натуральных чисел при вычислении на машине Тьюринга.**
- **На вход МТ подаются две последовательности десятичных чисел.**
- **Внешний алфавит МТ содержит десятичные цифры и пустой символ.**

- В качестве меры временной сложности T используется количество выполненных команд МТ для перехода из начальной конфигурации в заключительную.

- Емкостная сложность M вычислений на МТ определяется количеством ячеек ленты, которые заполнены непустыми символами либо посещались головкой во время работы МТ.

В общих чертах поведение МТ можно описать так

- Сначала головка перемещается к концу второй последовательности стирает младший разряд,
- затем перемещается к младшему разряду первого числа и заменяет его на сумму младших разрядов и т.д.

- Общее количество действий $T(n)=O(n^2)$ и задействованных ячеек $M(n)=O(n)$.
- При этом для получения одного разряда результата требуется порядка n действий.

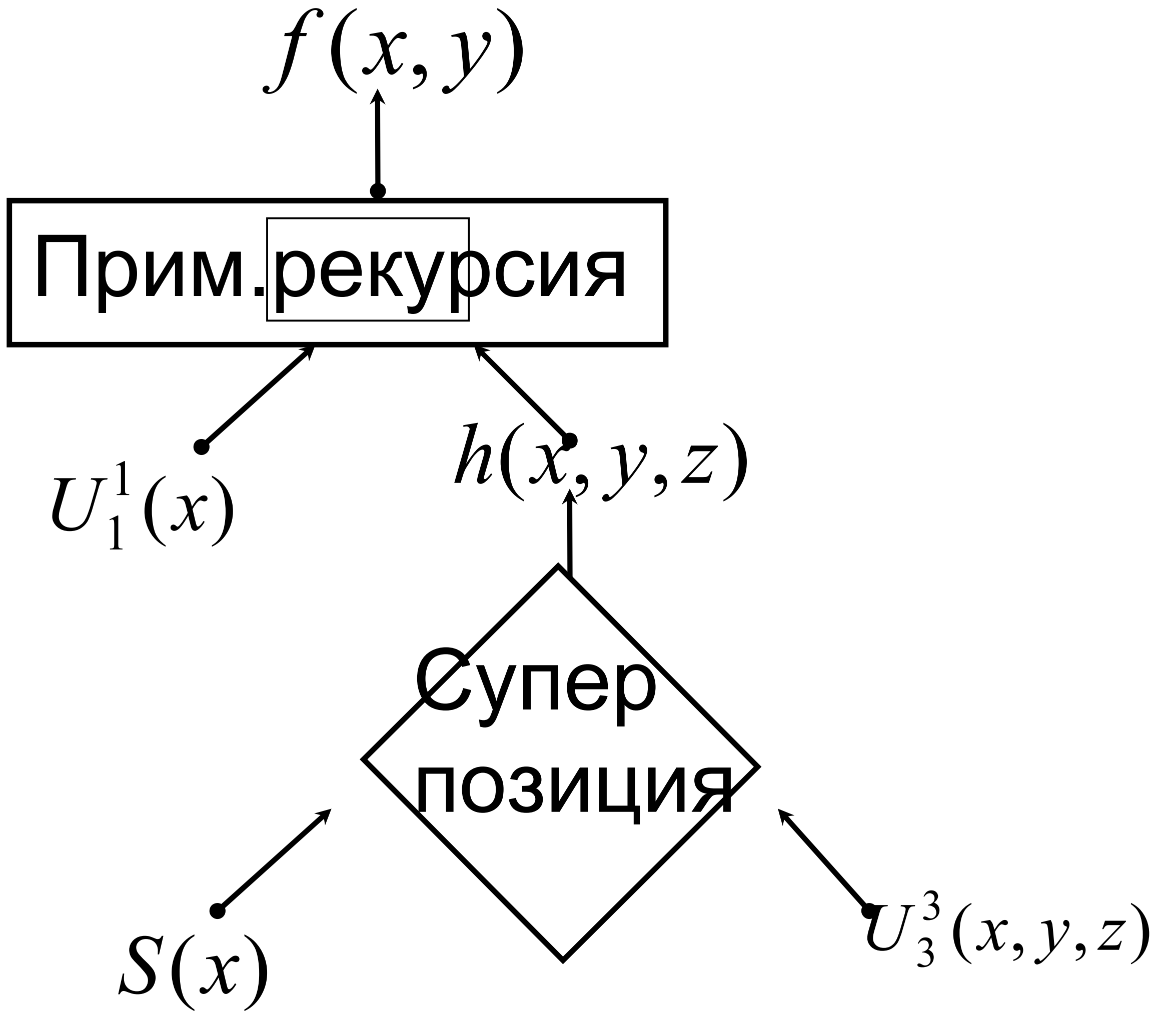
- **Таким образом, временные сложности алгоритма сложения столбиком и алгоритма сложения на МТ полиномиально эквивалентны.**

- **Арифметическая функция f(x,y)=x+y является примитивно рекурсивной, поскольку**

$$f(x,0) = x = U_1^1(x) = g(x)$$
$$f(x, y+1) = x + y + 1 =$$
$$= f(x, y) + 1 = S(f(x, y)) = h(x, y, f(x, y))$$

- **где** $h(x, y, z) = S(U_3^3(x, y, z))$

- **Поскольку функции *g* и *h* являются примитивно рекурсивными, то и функция *f* является примитивно рекурсивной.**
- **Процесс получения функции можно представить графически в виде дерева**


$f(x,y)$
Прим. рекурсия
$U_1^1(x)$
$h(x,y,z)$
Супер
позиция
$S(x)$
$U_3^3(x,y,z)$

- В качестве временной сложности можно использовать количество операций суперпозиции, примитивной рекурсии и минимизации, которые необходимы при получении функции.

- Количество вершин в дереве заменит емкостную сложность получения рекурсивной функции.
- В данном случае T=2, M=5.

8. Классы сложности P и NP

- Нижние оценки временной сложности алгоритма позволяют судить о том, насколько эффективен алгоритм.
- Однако получение прямых нижних оценок удается только в очень редких случаях.
- Кроме того, функции емкостной и временной сложности определяются для конкретных алгоритмических систем.

- Будем считать, что алгоритмы реализуемы на машине Тьюринга и сложность алгоритмов определяется в рамках алгоритмической системы Тьюринга.

- **Алгоритм, обе функции сложности которого полиномиальные, называется *полиномиальным алгоритмом.***

- **Алгоритм, у которого хотя бы одна из двух функций сложности экспоненциальная называется *экспоненциальным алгоритмом*.**

- **Теория сложности алгоритмов определяет классы алгоритмов по сложности.**

- Обозначим P– класс, содержащий все полиномиальные алгоритмы;

  E – класс, содержащий все экспоненциальные алгоритмы.
- Нетрудно видеть, что

  P строго содержится в E.

- **Задача, решаемая полиномиальным алгоритмом, называется *легкоразрешимой* задачей.**
- **Задача, которую нельзя решить полиномиальным алгоритмом называется *трудноразрешимой*.**

- В число трудных задач входят алгоритмически неразрешимые задачи.
- Неразрешимость есть крайний случай экспоненциальности.

- Полиномиальные алгоритмы обладают свойством замкнутости, т.е. можно комбинировать различные полиномиальные алгоритмы, используя один в качестве «подпрограммы» другого и при этом результирующий алгоритм будет иметь полиномиальную сложность.

- **Аналогичное замечание можно сделать и относительно экспоненциальных алгоритмов.**

- **Помимо полиномиальных и экспоненциальных алгоритмов существуют алгоритмы, не попадающие ни в один их этих классов.**

- Все алгоритмы, рассматриваемые до сих пор, были детерминированными, т.е. результат текущего шага алгоритма однозначно определял действия следующего шага.
- В недетерминированном алгоритме результат текущего шага алгоритма допускает более одной возможности для последующих шагов.

- Недетерминированные алгоритмы не являются разновидностью вероятностных или случайных алгоритмов, но за один такт работы такие алгоритмы могут выполнять несколько действий одновременно.

- Для моделирования таких алгоритмов используют недетерминированные машины Тьюринга, в которых одна конфигурация может иметь несколько исходов (ответов).

- Каждый ответ обрабатывается другой машиной или машина сама выбирает наилучший ответ
- Такая машина возможна, но требует экспоненциального размножения конструкции.

- Определим класс *NP* как класс всех задач, которые можно решить недетерминированными алгоритмами, работающими в течение полиномиального времени.
- Очевидно, $P \subseteq NP$

- Класс NP охватывает многие задачи: задача о выполнимости, задача коммивояжера, решение систем уравнений с целыми переменными, составление расписаний с определенными условиями, задача о рюкзаке, оптимальный раскрой и т.д.

- Все они решаются на детерминированных машинах Тьюринга экспоненциально.
- Они трудны, но не доказано, что их нельзя упростить.

- Если хотя бы одну из них можно решить полиномиально, то все другие также решаются полиномиально.
- Общие подзадачи NP-задач могут быть легкоразрешимыми.