Теория сложности вычислений

Хайруллин Альфред

Теория сложности вычислений

- **Проблема:**

Для некоторых алгоритмически разрешимых задач, никто не смог найти *эффективный* алгоритм.

- **Цель:**

Измерить объем компьютерных ресурсов, необходимых для решения конкретных задач. Классифицировать алгоритмические задачи. Какой ценой.

Вычислительная сложность

- ***Сложность*** – число компьютерных команд, выполняемых при вычислении алгоритма.

- Если команды выполняются последовательно – ***временная сложность*.**

- ***Память*** - число переменных или единиц памяти, используемых в алгоритме.

Пример подсчета вычислительной сложности

- Вычислить a x^2 + b x + c для значений a=3, b = 4, c = 5, x = 7
- Временная сложность – 5
- Память – 9

- Преобразуем задачу к виду (a x + b) x + c, получим временную сложность - 4

Упражнения

Пример:

- $x^{16} = x\ x\ x\ x\ x\ x\ x\ x\ x\ x\ x\ x\ x\ x\ x\ x$ или
- $x^{16} = (((x^2)^2)^2)^2$

Покажите как можно вычислить:

- x^6 используя 3 умножения
- x^{33} используя 6 умножений

Ценность измерения сложности вычислений

Если определить временную сложность алгоритма, то можно достоверно оценить время работы алгоритма по входным данным без выполнения соответствующих вычислений.

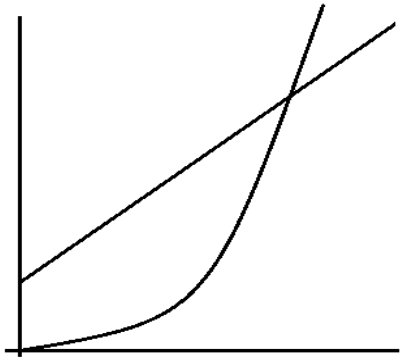
Пример: Посчитать средний возраст жителей некоторого поселка.

$$Time_A(n) = n + 1$$

Сравнение функций сложностей

- Рассмотрим две временные функции

  $Time_A(n) = 3n + 1$ и $Time_B(n) = n^2$



- Упражнение: Есть два алгоритма для задачи

  $Time_A(n) = 2\,n^2$ $Time_B(n) = 40\,n + 700$

Сравните для каких n какой алгоритм более выгоден.

Применимость алгоритмов.

- Пользователь знает точно, сколько времени ждать результата. Мы можем оценить сколько операций компьютер может сделать за приемлемое время для пользователя – допустим 10^{16}
- Если функция $3n + 1 < 10^{16}$, алгоритм А всегда применим
- Для функции $10^n < 10^{16}$ алгоритм А применим только для $n < 16$, что плохо

Пределы разрешимости.

- ***Сложность задачи U –*** это сложность самого лучшего (быстрого) алгоритма для U
- Пусть построен алгоритмом A для решения U. Тогда временная сложность $Time_A(n)$ является ***верхней границей*** **временной сложности U**.
- **f(n) –** ***нижняя граница*** **временной сложности U,** если не существует какого-либо алгоритма B для U с $Time_B(n) \leq f(n)$

f(n) \ n		10	50	100	300
10 n		100	500	1000	3000
$2 n^2$		200	5000	20000	180000
n^3		1000	125000	1000000	27000000
2^n		1024	16 цифр	31 цифра	91 цифра
n!		3600000	65 цифр	158 цифр	615 цифр

Практически разрешимые задачи

- Алгоритм A с $Time_A(n) \leq c\ n^d$ для некоторых постоянных c и d называется ***полиномиальным алгоритмом.***
- Каждая задача, которая может быть разрешима полиномиальным алгоритмом называется ***практически разрешимой***.
- **P** обозначает класс всех разрешимых задач принятия решения.

Сложные задачи (NP-трудные)

- Задача U является ***сложной***, если существование полиномиального алгоритма для U также предполагает существование полиномиальных алгоритмов для нескольких тысяч других задач, которые считаются сложными (т.е. для которых не могут найти эффективные алгоритмы)

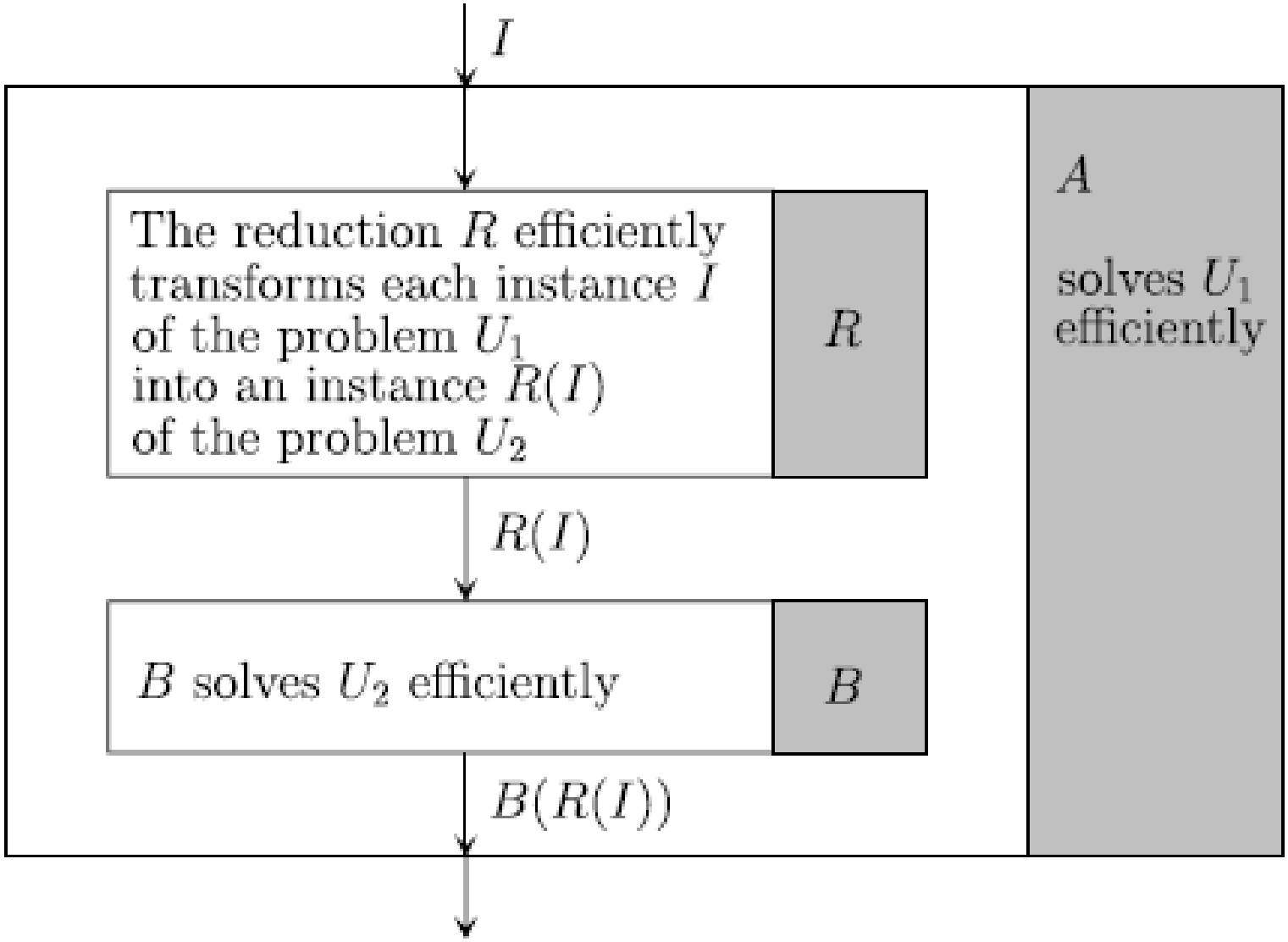
Сведение задач

Снова используем *метод редукции*

- Можно свести задачу U_1 к задаче U_2 , разработав эффективный алгоритм R, трансформирующий любой случай I задачи U_1 в эквивалентный случай R(I) задачи U_2

- Полиномиальная редукция, $U_1 \leq_{pol} U_2$

- Пример: $a + bx = 0$ и $a + bx = c + dx$


I
The reduction R efficiently transforms each instance I of the problem U_1 into an instance $R(I)$ of the problem U_2
R
$R(I)$
B solves U_2 efficiently
B
$B(R(I))$
A
solves U_1 efficiently

Что делать если ваша задача сложная

- Незначительное изменение в условиях задачи, может вызвать большое изменение в количестве компьютерной работы, необходимой для вычисления решения.

- Ищем близкое к оптимальному решение, вместо оптимального. Например с отклонением в 1%.

Спасибо за внимание