Теория алгоритмов: методы и результаты

С.М.Дудаков

2015–11–15

Теория алгоритмов

Теория алгоритмов — раздел математики, изучающий потенциальные возможности вычислительной техники и их ограничения

Теория алгоритмов

Теория алгоритмов — раздел математики, изучающий потенциальные возможности вычислительной техники и их ограничения

Неформально

Теория алгоритмов изучает что могут, а что не могут компьютеры

Теория алгоритмов

История

- Вычислительные устройства существуют уже несколько веков
- В 20–30 годы 20в. появляются вычислительные машины, способные исполнять программы
- Тогда теория алгоритмов и зарождается
- По сравнению с другими разделами математики — молодая наука

Теория алгоритмов

Что изучает

- Как моделировать вычислительные устройства, алгоритмы, языки программирования?
- Можно ли решить ту или иную проблему с помощью компьютера?
- Сколько ресурсов (времени, памяти) потребуется на решение задачи с помощью компьютера?

Основа

Главный принцип

Чтобы исследовать объект средствами математики нужно построить его математическую модель

Требования

- Сделать модель как можно проще
- Сохранить как можно больше свойств реальных объектов

Пример

Язык C

Почему его сложно исследовать?

Пример

Язык C

Почему его сложно исследовать?

Слишком богат

- Много операций
- Много конструкций
- Много типов данных

Пример

Язык C

Почему его сложно исследовать?

Слишком богат

- Много операций
- Много конструкций
- Много типов данных

Много ограничений

- Ограничения на числа

Как упростить?

Убрать ограничения

Предположим, что переменные целого типа `int` могут хранить сколь угодно большие натуральные числа

Что это дает?

Если что-то нельзя сделать в таком «обогащенном C», то это нельзя и в «обычном C».

Что делать дальше?

Убрать лишние типы данных

- Действительное число — пара натуральных
  $x = m/n$,
  например, $0.23 \mapsto (23, 100)$
- Строка — массив кодов символов,
  например, «abc» $\mapsto (65, 66, 67)$
- Массив целых чисел $(a_1, a_2, \ldots, a_n)$ — число
  $p_1^{a_1} p_2^{a_2} \ldots p_n^{a_n}$,
  например, $(2, 3, 1) \mapsto 2^2 \times 3^3 \times 5^1 = 540$
- И т.д.
- Останутся только целые числа типа `int`

Что еще?

Убрать лишние операции

- Умножение: вместо `x = y * z;` можно написать
- ```
  x = 0; u = z;
  while(u) {
    u--; x = x + y; }
  ```
  ```

Что еще?

Убрать лишние операции

- Умножение: вместо `x = y * z;` можно написать

- ```
  x = 0; u = z;
  while(u) {
    u--; x = x + y; }
  ```

- Сложение: вместо `x = y + z;` можно написать

  ```
  x = y; u = z;
  while(u) {
    u--; x++; }
  ```
  ```

Что еще?

Убрать лишние операции

- Присваивание: вместо `x = y;` можно написать
- ```
  u = 0; x = 0;
  while(y) {
    y--; u++; }
  while(u) {
    u--; x++; y++; }
  ```
  ```

Что еще?

Убрать лишние операции

- Присваивание: вместо `x = 0;` можно написать
- ```
  while(x)
    x--;
  ```
  ```

Что еще?

Убрать лишние операции

- Присваивание: вместо `x = 0;` можно написать
- ```
  while(x)
    x--;
  ```

В результате

Останутся только операции инкремента `++` и декремента `--`
  ```

Дальше?

Убрать лишние конструкции

- Заменяем

```
for(a; b; c)
  0
```

- на

```
a;
while(b) {
  0
  c;
  }
```

Дальше?

Убрать лишние конструкции

- Заменяем

```
while(e)
  O
```

- на

```
   goto b;
a: O
b: if(e) goto a;
```

Дальше?

Убрать лишние конструкции

- Заменяем

```
if(e) O1
else O2
```

- на

```
if(e) goto a;
O2 goto b;
a: O1
b:
```

Что останется?

Очень простой язык

- `x++;`
- `x--;`
- `goto a;`
- `if(x) goto a;`

Можно эффективно изучать математическими средствами

Счетчиковая машина

М.Л.Минский

Программа с операторами вида

- `a: x++; goto b;`
- `a: if(x) x--; goto b; else goto c;`

Нормальный алгорифм

А.А.Марков

Множество инструкций вида

$\alpha \mapsto \beta$, где α и β — слова

Выполнение

Последовательная замена левых слов правыми, до тех пор, пока хоть одна замена возможна

Нормальный алгорифм

Пример

Алгорифм вида $a \mapsto bb, b \mapsto cc$ преобразует слово из n букв a в слово из $4n$ букв c

Выполнение

$aa \mapsto bba \mapsto ccba \mapsto cccca \mapsto ccccbb \mapsto ccccccb \mapsto cccccccc$

Клеточный автомат

Дж.фон Нейман

Множество инструкций вида

$abc \mapsto d$, где a, b, c, d — некоторые символы

Выполнение

- Имеется бесконечная лента разбитая на ячейки, в которых написаны символы:

$\cdots$	a	c	a	b	c	d	b	$\cdots$

Клеточный автомат

Выполнение

- Команда $abc \mapsto d$ означает, что если в некоторой ячейке записано b, слева от нее — a, а справа — c, то нужно b заменить на d
- Замены во всех ячейках выполняются одновременно и синхронно

Клеточный автомат

Пример

- Пусть есть команды $aba \mapsto a$, $baa \mapsto a$, $aab \mapsto b$
- Тогда за один шаг лента

$\cdots$	a	b	a	a	b	a	a	$\cdots$

превратится в

$\cdots$	a	a	b	a	a	$\cdots$

- А затем в

$\cdots$	b	a	a	$\cdots$

Игра «Жизнь»

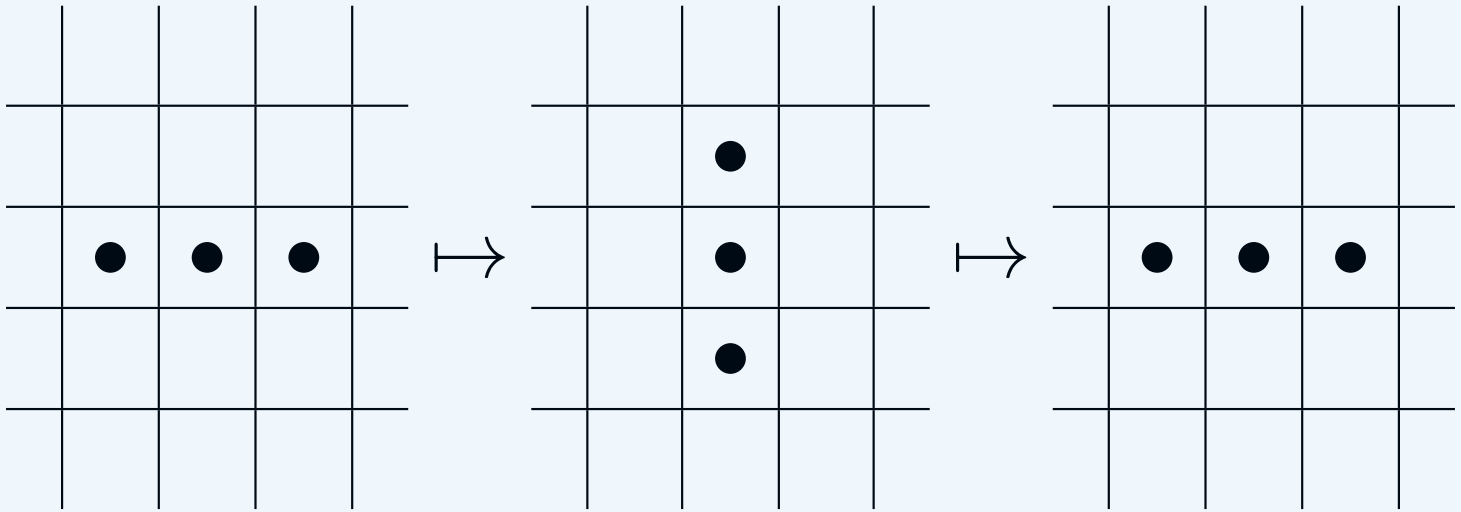
Дж.Н.Конвей, http://www.conwaylife.com/

Правила

- Двумерное бесконечное поле, разбитое на клетки, в которых могут находится фишки
- Если у фишки меньше двух или больше трех соседей, то она снимается
- Если у пустой клетки ровно три соседа, то на нее ставится новая фишка

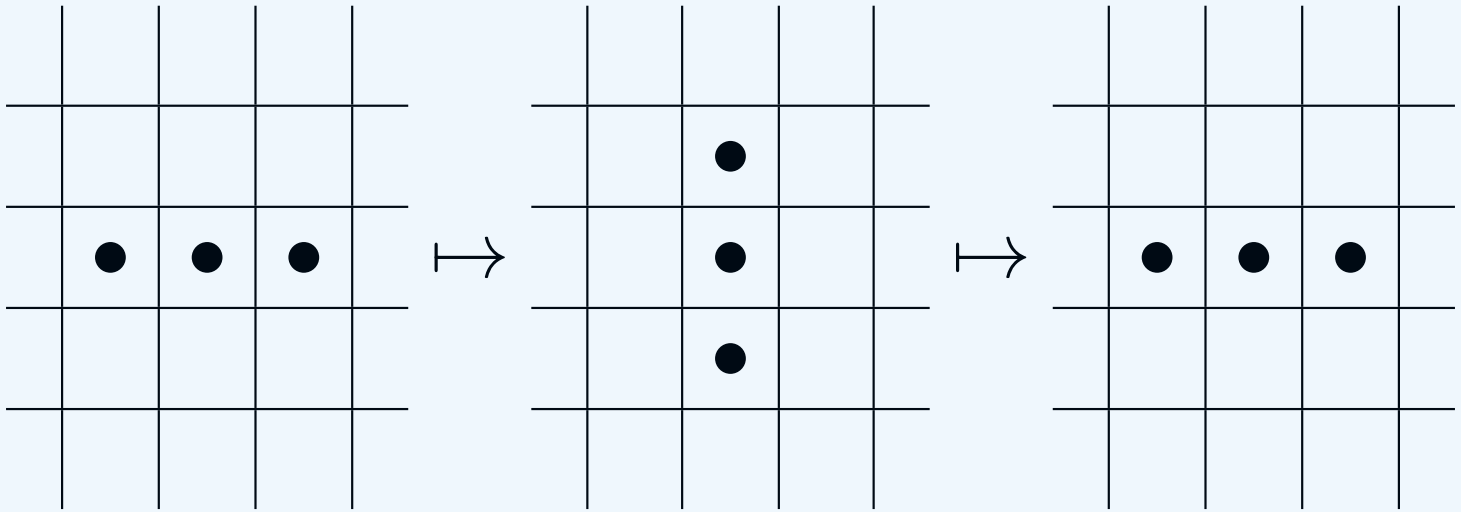
Игра «Жизнь»

Пример: «Мигалка»



Игра «Жизнь»

Пример: «Мигалка»



Полноценный компьютер

Существуют конфигурации фишек в игре «Жизнь», которые позволяют промоделировать работу любого вычислительного устройства

Эквивалентность моделей

Все рассмотренные модели эквивалентны

Любой алгоритм, который можно реализовать на одной из них, можно реализовать во всех остальных

Тезис Черча-Тьюринга

- Любой алгоритм можно реализовать в каждой из перечисленных моделей
- Более чем за 80 лет развития теории алгоритмов этот тезис не опровергнут

Алгоритмические проблемы

Задача

По строке (набору строк или чисел) S определить, обладает ли она нужным нам свойством или нет?

Примеры

- По строке S определить, является ли она программой на языке C?
- По строке S определить, является ли она программой на языке C, которая никогда не зацикливается?

Алгоритмические проблемы

Примеры

- Ответы «нет», «нет»:

```
void f(int x) { return y; }
```

- Ответы «да», «нет»:

```
void f(int x) { while(x); }
```

- Ответы «да», «да»:

```
void f(int x) { return x; }
```

Разрешимые и неразрешимые проблемы

Разрешимость

Проблема (алгоритмически) разрешима, если существует программа, которая всегда останавливается и выдает правильный ответ на вопрос

Неразрешимость

Если такой программы нет — проблема (алгоритмически) неразрешима

Разрешимые и неразрешимые проблемы

Примеры

- Определить, является ли строка S программой на языке C алгоритмически возможно — проблема разрешима
- Определить, является ли строка S программой на языке C, которая никогда не зацикливается, алгоритмически нельзя — проблема неразрешима

Проблема самоприменимости

Формулировка

По алгоритму

```
void f(const char * s) {...}
```

определить, не «зациклится» ли он, если в качестве аргумента `s` подать ее же текст.

Самоприменимость

Алгоритм

```
void f(const char * s) {...}
```

самоприменим, если он при указанном условии не зациклится

Самоприменимость

Примеры

- Самоприменим:

```
void f(const char * s) { return; }
```

- Не самоприменим:

```
void f(const char * s)
{ while(strlen(s) > 10); }
```

Проблема самоприменимости

Утверждение

Проблема самоприменимости алгоритмически неразрешима

Проблема самоприменимости

Утверждение

Проблема самоприменимости алгоритмически неразрешима

Доказательство от противного

- Допустим, что можно написать алгоритм
  `int self(const char * s) {...}`
  который при подаче на вход текста алгоритма
  `void f(const char * s) {...}`
  возвращает 1, если `f` самоприменим, или 0, если нет

Проблема самоприменимости

Напишем такой алгоритм

```
int self(const char * s) { ... }
void f(const char * s)
{ while(self(s)); }
```

Алгоритм `f` зациклится, если `self` возвращает 1, или остановится, если `self` возвращает 0

Алгоритм `f` или самоприменим, или нет

Проблема самоприменимости

Допустим, f самоприменим

- Алгоритм `self` вернет 1 при подаче на вход текста `f`
- Тогда `f` зациклится при подаче на вход текста `f`
- То есть `f` несамоприменим
- Противоречие

Проблема самоприменимости

Допустим, f несамоприменим

- Алгоритм `self` вернет 0 при подаче на вход текста `f`
- Тогда `f` остановится при подаче на вход текста `f`
- То есть `f` самоприменим
- Противоречие

Проблема самоприменимости

Итог

- В любом случае приходим к противоречию
- Значит, исходное предположение неверно
- Следовательно, такого алгоритма `self` быть не может

Проблема самоприменимости

Итог

- В любом случае приходим к противоречию
- Значит, исходное предположение неверно
- Следовательно, такого алгоритма `self` быть не может

Проблема самоприменимости алгоритмически неразрешима

Проблема остановки

Формулировка

По алгоритму

```
void f(const char * s) {...}
```

и строке S определить, не «зациклится» ли `f`, если в качестве аргумента подать строку S

Проблема остановки

Утверждение

Проблема остановки алгоритмически неразрешима

Проблема остановки

Утверждение

Проблема остановки алгоритмически неразрешима

Метод сведения

- Возьмем в качестве S текст алгоритма `f`
- Тогда проблема остановки превратится в проблему самоприменимости
- Если бы мы смогли решить проблему остановки, то смогли бы решить и проблему самоприменимости.

Проблема остановки

Проблема остановки алгоритмически неразрешима

Проблема остановки

Проблема остановки алгоритмически неразрешима

Невозможно автоматически проверять правильность программного обеспечения

Проблема остановки

Проблема остановки алгоритмически неразрешима

Невозможно автоматически проверять правильность программного обеспечения

Правильность программного обеспечения может контролироваться только человеком

Функция «Усердного бобра»

Определение функции BB (Busy Beaver)

- Среди всех алгоритмов вида
  `int f() {...}`
  длина которых не превосходит n символов, существует такой, который возвращает наибольший результат: X.
- Тогда $BB(n) = X$
- Если заменить `return e;` на `return e+1;`, то новая функция будет возвращать большее значение, поэтому $BB(n+2) > BB(n)$

Функция «Усердного бобра»

Утверждение

Функция «Усердного бобра» алгоритмически невычислима

Функция «Усердного бобра»

Утверждение

Функция «Усердного бобра» алгоритмически невычислима

Доказательство от противного

- Предположим, что можно написать алгоритм `int bb(int n) {...}` вычисляющий функцию «Усердного бобра»

Функция «Усердного бобра»

Напишем такой алгоритм

```
int bb(int n) { ... }
void f()
{ int n = 2; n += 6; ...; n += 6;
  return bb(n); }
```

Здесь имеется C операторов `n += 6;`, где C — длина всех остальных частей алгоритма

Функция «Усердного бобра»

Доказательство от противного

- Длина алгоритма `f` равна $C + 5C = 6C$
- После выполнения операторов сложения значение переменной `n` будет равно $6C + 2$
- Мы получили алгоритм длины $6C$, который возвращает такое же значение, как алгоритм `bb` на аргументе $6C + 2$
- Но $BB(6C + 2) > BB(6C)$, поэтому алгоритм `bb` не может вычислять функцию BB

Сжатие информации

Архивация

По исходной строке S построить ее сжатое представление (архив) A

Разархивация

По сжатому представлению A восстановить исходную строку S

Математическая модель

- Архив — программа
- Разархивация — выполнение программы и получение нужного результата

Сжатие информации

Пример

- Строка состоящая из миллиарда символов `X`
- Можно получить так:

```
for(int i = 0; i < 1000000000; i++)
  putchar('X');
```

Проблема оптимального сжатия

Формулировка

По строке S найти алгоритм (архив)

```
void f() {...}
```

наименьшей длины, который генерировал бы S

Проблема оптимального сжатия

Формулировка

По строке S найти алгоритм (архив)

```
void f() {...}
```

наименьшей длины, который генерировал бы S

Утверждение

Не существует алгоритма, который по строке S определял бы размер ее наименьшего архива

Проблема оптимального сжатия

Формулировка

По строке S найти алгоритм (архив)

`void f() {...}`

наименьшей длины, который генерировал бы S

Утверждение

Не существует алгоритма, который по строке S определял бы размер ее наименьшего архива

Доказательство от противного

Допустим, что такой алгоритм есть:

```
int arch(const char * s) { ...}
```

Невозможность оптимального сжатия

Напишем такой алгоритм

```
int arch(const char * s) { ... }
const char * next(const char * s) { ... }
void f()
{ int n = 1; n += 6; ...; n += 6;
  const char * s = "";
  while(arch(s) < n) s = next(s);
  puts(s); }
```

C операторов `n += 6;`, где C — количество символов во всей остальной программе

Невозможность оптимального сжатия

Результат

- Длина алгоритма `f` равна $6C$
- Значение переменной `n` после первой строки будет равно $6C + 1$
- Алгоритм `f` напечатает строку, которую нельзя сжать сильнее чем до $6C + 1$, но сам имеет длину всего $6C$
- Противоречие

Заключение

Итоги

- Построение математических моделей вычислительных устройств
- Неразрешимость проблем самоприменимости, остановки, «Усердного бобра» и оптимального сжатия

Конец

Спасибо за внимание!

Вопросы, пожалуйста. . .