

Адаптивное арифметическое кодирование

Для арифметического кодирования существует адаптивный алгоритм. Его идея заключается в том, что частоты символов, используемые при кодировании входной последовательности изменяются по мере получения каждого нового символа. То есть происходит «адаптация» алгоритма к входным данным. При декодировании происходит аналогичный процесс.

Алгоритм кодирования

На вход алгоритму передаётся последовательность символов и алфавит. Каждому символу алфавита $\alpha \in \Sigma$ сопоставляется его вес w_α . В начале работы алгоритма все веса символов равны 1. Символы располагаются в естественном порядке, например по возрастанию. Вероятность каждого символа α — $p(\alpha)$ устанавливается равной его весу, делённому на суммарный вес всех символов: $p(\alpha) = \frac{w_\alpha}{\sum_{i=1}^n w_i}$. После получения очередного символа и построения нужного интервала, вес символа увеличивается на 1. Когда все символы последовательности будут обработаны, необходимо либо записать маркер конца последовательности, либо запомнить её длину, чтобы позже передать декодировщику. После получения нужных границ интервала необходимо выбрать число x описывающее кодирование: $x \in [l, r]$. Выбор числа x производится также как и в неадаптивном алгоритме. Выбирается число вида $\frac{x}{2^p} : x, p \in \mathbb{N}$.

Псевдокод алгоритма

- **in** — строка, подаваемая на вход;
- **Segment** — структура, задающая подотрезок отрезка $[0, 1)$, соответствующая конкретному символу. Имеет следующие поля:
 - **left** — левая граница подотрезка
 - **right** — правая граница подотрезка
- **m** — мощность алфавита;

- `weight` — веса символов алфавита;
- `segments` — набор подотрезков, соответствующих символам алфавита;
- `left`, `right` - границы отрезка, содержащие возможный результат арифметического кодирования;

Подотрезок

```
1 struct Segment:
2     double left;
3     double right;
```

Определение начальных границ подотрезков

```
1 Map<char, Segment> defineSegments(set<char> alphabet):
2     double p = 1 / alphabet.size()
3     Segments[m] segments
4     double curLeft = 0
5     double curRight = p
6     for i = 0 to m - 1:
7         segments[i].left = curLeft
8         segments[i].right = curRight
9         curLeft = curRight
10        curRight = curRight + p
11    return segments
```

Перестройка подотрезков

```
1 Map<Char, Segment> resizeSegments(alphabet : char[m], weight :
2 Map<char, int>, segments : Map<char, Segment>):
3     double l = 0
4     int sum = 0
5     for i = 0 to m - 1:
6         sum = sum + weight[i]
7     for i = 0 to m - 1:
8         char c = alphabet[i]
9         segments[c].left = l
```

```
10         segments[c].right = 1 + (weight[c] / sum)
11         l = segments[c].right;
12     return segments
```

Построение алфавита по входной строке

```
1  set<char> getAlphabet(in : String):
2      Set<char> alphabet
3      for i = 0 to in.length() - 1:
4          alphabet.add(in[i])
5      return alphabet
```

Определение начальных весов символов алфавита

```
1  Map<char, int> defineWeights(alphabet : Set<char>):
2      Map<char, int> weight
3      for i = 0 to m - 1:
4          char c = alphabet[i]
5          weight[c] = 1
6      return weight
```

Кодирование строки

```
1  double adaptiveCoding(in : String, alphabet : Set<char>):
2      int[m] weight = defineWeights(alphabet)
3      int[m] segments = defineSegments(alphabet)
4      double left = 0
5      double right = 1
6      for i = 0 to n - 1:
7          char c = alphabet[i]
8          weight[c]++
9          double newRight = left + (right - left) * segments[c].right
10         double newLeft = left + (right - left) * segments[c].left
11         left = newLeft
12         right = newRight
13         resizeSegments(alphabet, weight, segments)
14     return (left + right) / 2;
```

Алгоритм декодирования

При декодировании последовательности символов также используется множество весов w символов алфавита Σ . В начале работы алгоритма все веса символов равны 1. На каждом шаге определяется интервал, содержащий данный код, по интервалу находится символ, который потом записывается в выходную последовательность. Вес полученного символа α увеличивается на 1. Отрезки соответствующие символам алфавита перестраиваются в зависимости от изменённых весов символов и размера текущего подотрезка. При получении символа конца последовательности или обработки нужного числа символов алгоритм завершает работу.

Псевдокод алгоритма

При декодировании строки будут использоваться функции `defineWeights()` и `defineSegments()` из алгоритма кодирования.

Декодирование

- `code` — вещественное число, подаваемое на вход;
- `len` — длина декодируемой строки;
- `m` — мощность алфавита;
- `weight` — веса символов алфавита;
- `segments` — набор подотрезков, соответствующих символам алфавита;

```
1  String decode(code : double, alphabet : Set<char>, int len):
2      Map<char, int> weight = defineWeights(alphabet)
3      Map<char, Segment> segments = defineSegments(alphabet)
4      String ans = ""
5      for i = 0 to len - 1:
6          for j = 0 to m - 1:
7              char c = alphabet[j]
8              if code >= segments[c].left and code < segments[c].right:
9                  ans = ans + c
10                 weight[c]++
11                 code = (code - segments[c].left) /
12 segments[c].right - segments[c].left)
13                 resizeSegments(alphabet, weight, segments)
14                 break;
15     return ans
```

Необходимость применения адаптивного алгоритма возникает в том случае, если вероятностные оценки символов сообщения не известны до начала работы алгоритма. Преимущество такого подхода кодирования заключается в том, что декодировщику не нужно передавать вероятностные оценки для символов, он будет строить их по мере декодирования сообщения, что может сильно сократить вес такого сообщения.