Лекция 11

Алфавит, слово и язык в программировании

Для обработки информации мы представляем данные (объекты обработки) как последовательности символов. Мы фиксируем для представления данных специальный набор символов – *алфавит.*

Определение: Алфавитом называется любое *непустое конечное множество*. Каждый элемент алфавита называется символом.

Для представления нужных нам объектов мы можем выбрать любой алфавит, содержащий конечное число символов.

Примеры:

Σ_{bool} ={0,1} –булевский алфавит

$\Sigma_{лат}$ ={a,b,c,d,….,z} –латинский алфавит

$\Sigma_{keyboard}$ = $\Sigma_{лат}$ ∪ {A,B,C,…,Z , >,<,(,), …..,!} – алфавит всех символов на клавиатуре

Σ_m ={0,1,2,….,m-1} для каждого m >=1 – алфавит для записи чисел в m-ичной системе счисления.

Здесь N={1,2,3,…….} –множество натуральных чисел

N_0=N ∪ {0}

Определим слово как некоторую последовательность символов. Слово в программировании означает произвольный текст.

Определение. Пусть Σ - некоторый алфавит. Слово над Σ - любая конечная последовательность символов алфавита Σ.

Длина слова w над Σ, обозначаемая как $|w|$, есть число символов в этом слове.

Пустое слово λ - единственное слово, состоящее из нулевого количества символов. ($|\lambda|=0$)

Σ^* - множество всех слов над алфавитом Σ.

$\Sigma^+=\Sigma^*-\{\lambda\}$ - означает множество всех слов за исключением пустого. (74)

Символ пробела над алфавитом клавиатуры отличен от λ, так как пробел – элемент алфавита Σ_{keyboard}, то $|\text{пробел}|=1$.

Примеры

1. $(\Sigma_{bool})^*=\{\lambda,0,1,00,01,10,11,000,001,010,100,011,\ldots..\}$
$=\{\lambda\} \cup \{x_1 x_2 \ldots\ldots x_i \mid i \in N_0, x_j \in \Sigma_{bool}$ для $j=1,\ldots,i\}$

Мы видим, что существует возможность перечисления всех слов над заданным алфавитом. При этом слова записываются в порядке возрастания их длин, то есть одно слово за другим для каждой рассматриваемой длины.

2.Дан двухбуквенный алфавит $\Sigma^2=\{a,b\}$

(Σ^{+2}) - это множество слов, состоящих из двух букв, прописанных в произвольном порядке за исключением пустого.
Например, множество всех слов длины 2 :

$\Sigma^{+2}=\{aa, bb, ab, ba\}$; (75)

Упражнения:

1 а) Дан двухбуквенный алфавит $\Sigma^2=\{a,b\}$.
Напишите множество всех слов длины 3, прописанных в произвольном порядке за исключением пустого. (Σ^{+3}) (76)

$\Sigma^{+3}=\{aaa, aab, aba, abb, baa, bab\ bba, bbb\}$

б) Покажите, как подсчитать, сколько слов длины 3 существует над заданным алфавитом.

$\Sigma^{+3}=8$

в) Вычислите номер слова aba в заданном алфавите

??

г) Найдите слово в данном двухбуквенном алфавите, имеющим номер 5.

??

2 Пусть дан язык $\Sigma_1=\{a^n b^m\}$; n=1,2….., m=1,2….
Определить, принадлежит ли слово *aabbb* данному языку.

??

А слово bab?

??

3 Пусть дан язык $\Sigma_2=\{a^n b^n\}$; n=1,2…..
Определить, принадлежит ли слово aaaabbbb данному языку?

??

А слово abab?

??

А слово bbaa? (77)

??

Слова можно использовать для представления различных объектов – это могут быть, например, числа, формулы, графы, деревья, программы.

Слово $x = x_1 x_2 \ldots x_n \in (\Sigma_{bool})^*$, $x_i \in \Sigma_{bool}$ для i=1,….,n можно рассматривать как двоичное представление неотрицательного числа

$$Number(x) = \sum_{i=1}^{n} 2^{n-i} * x_i \qquad (2^0*1+2^1*0+2^2*1+\ldots.)$$

Наоборот, для любого неотрицательного целого числа m запись

$$Bin(m) \in (\Sigma_{bool})^*$$

обозначает наиболее короткое двоичное представление числа m. (Наиболее короткое означает, что первый символ должен быть равен единице). Поэтому

$$Number(Bin(m))=m.$$

Как определить зеркальный язык?

$\Sigma_4 = \{aa^{-1}\}$, где $a \in \Sigma^+$

Слова $abba \in \Sigma_4$, $baaaab \in \Sigma_4$,

а слова aba и aaa не принадлежат Σ_4.

Вывод:

Мы видим, что любой язык над алфавитом Σ суть некоторое n-арное отношение на множестве всех слов Σ^+, где n=1,2…… (арность – это размерность)

Давайте представим булевскую формулу с помощью операторов отрицания, конъюнкции и дизъюнкции ($\neg \wedge \vee$). Для этого будем использовать алфавит

$$\Sigma_{\text{лог}} = \{0,1,x,(,),\ \neg \wedge \vee\}$$

где x –булевы переменные, используемые как символы алфавита ($x_1, x_2, \ldots..$) и кодировка булевской переменной x_i как слова

xBin(i) для всех $i \in N$.

Все остальные символы в формуле переписываются в ее представлении один к одному. Тогда формула

$$(x_1 \vee x_7) \wedge \neg (x_{12}) \wedge (x_4 \vee x_8 \vee \neg(x_2))$$

имеет следующее представление:

$$(x1 \vee x111) \wedge \neg(x1100) \wedge (x100 \vee x1000 \vee \neg(x10)).$$

Полезная операция над словами – простая *конкатенация* двух слов.

Определение. Пусть Σ - некоторый алфавит. Конкатенация относительно Σ - это отображение $K : \Sigma^* \times \Sigma^* \to \Sigma^*$, заданное в виде

$$K(x,y)=x \circ y=xy$$

для всех $x, y \in \Sigma^*$

Пример

Пусть x=0aa1bb и y=111b для Σ ={0,1,a,b}.

Тогда K(x,y) = $x \circ y=xy$=0aa1bb111b. (78)

Для всех $x,y \in \Sigma^*$,

$$| xy | = |x \circ y| = |x| + |y|$$

Часто пишут xy вместо K(x,y) и $x \circ y$. (мощность равна сумме мощностей)

Определение.

Пусть Σ - некоторый алфавит. Для каждого $x \in \Sigma^*$ и каждого натурального i определим i-ю итерацию x^i слова x как

$$x^i = xx^{i-1},$$

где $x^0 = \lambda$.

Пример

$K(aabba, aaaaa) = aabbaaaaaa = a^2 b^2 a^6 = a^2 b^2 (aa)^3$

Введенное обозначение позволяет нам находить более короткое представление некоторых слов.

Определим подслова слова x как связанные части этого слова.

Связанные части слова

ab….. bb……..c ac……..b

_____

подслово

ab…….b b…….a

_____

префикс

a……b bb…ab

_____

суффикс

Определение. Пусть $v,w \in \Sigma^*$ для некоторого алфавита Σ. Тогда:

- v- подслово слова w $\Leftrightarrow \exists x,y \in \Sigma^* : w=xvy;$
- v- суффикс слова w $\Leftrightarrow \exists x \in \Sigma^*: w=xv;$
- v- префикс слова w $\Leftrightarrow \exists y \in \Sigma^*: w=vy.$

Определение. Пусть $x \in \Sigma^*$ и пусть $a \in \Sigma$ для некоторого алфавита Σ. Определим $|x|_a$ как число вхождений символа a в x.

(посмотреть множества – мощность и множество всех подмножеств)

Пример

$|(abbab)|_a = 2$ и $|(11bb0)|_0 = 1$

Для каждого $x \in \Sigma^*$ $|x| = \sum_{a \in \Sigma} |x|_a$

Для практической работы с языками надо помнить, что в принципе *языки суть множества*, поэтому для них можно использовать стандартные операции объединения и пересечения множеств. Сюда можно добавить *конкатенацию и звезду Клини.*

Определение. Язык над алфавитом Σ – это произвольное подмножество слов L из Σ^*. Если множество L конечно, то и язык, определяемый этим множеством, называется конечным.

Язык может быть бесконечным. Например, совокупность записей всех четных чисел в 10 –ой СС представляет собой бесконечный язык. Множество всех слов русского алфавита (но не русского языка!) , заканчивающихся гласной буквой, тоже бесконечный язык.

Язык L над алфавитом Σ называется универсальным, если $L=\Sigma^*$.

Пусть L_1 и L_2 – языки над алфавитом Σ . Тогда:

- *Объединением языков* L_1 и L_2 называется язык, обозначаемый $L_1 \cup L_2$,слова которого принадлежат по меньшей мере одному из этих языков.
- *Пересечением языков* L_1 и L_2 называется язык, обозначаемый $L_1 \cap L_2$, слова которого принадлежат каждому из этих языков.
- *Разность L_1 и L_2* – $L_1 \setminus L_2$, слова которого принадлежат L_1, но не принадлежат L_2.
- *Дополнение* L^C языка L относительно Σ - это язык Σ^* – L.

$L_0 = 0$ – пустой язык
(Язык L называется пустым, если множество L пусто.)

$L\lambda = \{\lambda\}$ –язык, содержащий единственное слово - пустое слово. (80)

Если L_1 и L_2 - языки над алфавитом Σ, то $L_1 \circ L_2 = L_1L_2 = \{ vw \mid v \in L_1, w \in L_2\}$
Это ***конкатенация*** языков L_1 и L_2. (объединение (сцепка) языков)
Если один из языков - пустой, то пустым является и язык $L_1 \circ L_2$.

Пусть L – некоторый язык над алфавитом Σ. Определим

$L^0 := L\lambda$, (нулевая степень любого множества неизменна.)

$L^{i+1} = L^i * L$ для всех $i \in N_0$, (остальные степени определяются рекурсивно , то есть $L^1 = L$; $L^2 = LL$)

$L^* = \bigcup_{i \in N_0} L^i$ - ***звезда Клини*** языка L (в сущности – это итерация, которая получается объединением всех неотрицательных степеней языка L.)

$L^+ = \bigcup_{i \in N} L^i = L*L^*$ (L^+ обозначает язык, из которого изъяли пустое слово)

Звезда Клини – это множество всех строк конечной длины, порожденное элементами множества L. Это унарная операция над множеством строк (или символов). Клини ввел это для описания некоторого автомата.

Примеры языков над алфавитом Σ={a,b}:

L1=0;
L2={λ};
L3= {λ,ab,abab};
L4=Σ^* = {λ,a,b,aa,bb,…..};
L5= Σ^+ ={a,b,aa,bb,….};
L6={a}* ={λ,a,aa,aaa,……}={a^i : i $\in$ N_0}
L7= Σ^3 ={aaa,aab,aba,abb,baa,bab,bba,bbb}

Все грамматически правильные тексты (английские) и множество всех синтаксически правильных программ на ЯВУ – всегда некоторый язык над алфавитом $\Sigma_{keyborard}$. (81)

Алгоритмические проблемы.

(*Проблема принадлежности*,
проблема существования Гамильтонова цикла (НС),
проблема выполнимости,
оптимизационная проблема)

Если рассматривать алгоритм как программу, то для решения алгоритмических проблем нам безразличен выбор конкретного языка программирования. Мы просто хотим, чтобы программа вычисляла правильный выход для любого входа.

Таким образом, алгоритм можно рассматривать как программу, заканчивающую работу для любого входа (то есть не имеющую бесконечных вычислений) и решающую данную проблему. Тогда, любая программа (алгоритм) выполняет отображение

A : Σ^*_1 -> Σ^*_2 для некоторых алфавитов Σ_1 и Σ_2.
Это значит, что:

- входы представлены как слова над алфавитом Σ_1;
- выходы представлены как слова над алфавитом Σ_2;
- “A ‘ однозначно определяет выход по каждому входу.

Для некоторого алгоритма A и входа x обозначим записью A(x) выход алгоритма A для этого входа. Два алгоритма (программы) A и B эквивалентны, если они работают над одним и тем же алфавитом Σ, и при этом A(x) =B(x) для всех $x \in \Sigma^*$.

Проблема принадлежности слова языку.

Проблема в том, чтобы по заданному языку L в алфавите Σ и по произвольному слову “a “ из Σ^*, определить, принадлежит ли слово языку L.

Сложность решения проблемы зависит от способа задания языка. Например, сложно определить по заданному в десятичной СС целому положительному числу, является ли оно простым.

Если рассматриваемая проблема принадлежности имеет *решающий алгоритм*, то говорят, что она *алгоритмически разрешима*.

// Но есть формализовано определенные языки, для которых //проблема принадлежности неразрешима.

Если проблема принадлежности имеет решающий алгоритм, то его сложность можно охарактеризовать зависимостью числа

выполняемых элементарных операций от длины тестируемого слова “a”.

Есть *класс регулярных языков*, которые характеризуются тем, что *для языков этого класса существуют решающие алгоритмы*, сложность которых линейно зависит от длины тестируемого слова.

Определение: для заданных алфавита Σ и языка $L \subseteq \Sigma^*$ проблема принадлежности (Σ, L) заключается в том, что для любого $x \in \Sigma^*$ необходимо ответить на вопрос, какое именно условие из следующих двух выполнено :

$x \in L$ или $x \notin L$.

Алгоритм A решает проблему принадлежности (L, Σ), если для всех $x \in \Sigma^*$ выполнено следующее:

$$A(x) = \begin{cases} 1, & \text{если } x \in L \\ 0, & \text{если } x \notin L. \end{cases}$$

тогда говорят, что A распознает язык L.

Если есть алгоритм для некоторого языка L, который распознает L, то говорят, что язык L принадлежит специальному классу функций, называемых *рекурсивными.*

Числовые функции, значения которых можно вычислить посредством алгоритма, называются *вычислимыми функциями*.

Тезис Черча говорит о том, что *класс рекурсивных функций тождественен классу всюду определенных вычислимых функций.*

Поэтому вопрос о вычислимости функции равносилен вопросу о ее рекурсивности.

Если вам удалось доказать, что решающая конкретную задачу функция не может быть рекурсивной, то следовательно, не существует и алгоритма решения этой задачи!

Проблема принадлежности (Σ, L)описывается так:

(Σ, L)

Вход : $x \in \Sigma^*$.

Выход : $A(x) \in \Sigma_{bool}=\{0,1\}$, где

$A(x)=$ 1,если $x \in L$ (да)
0, если $x \notin L$ (нет)

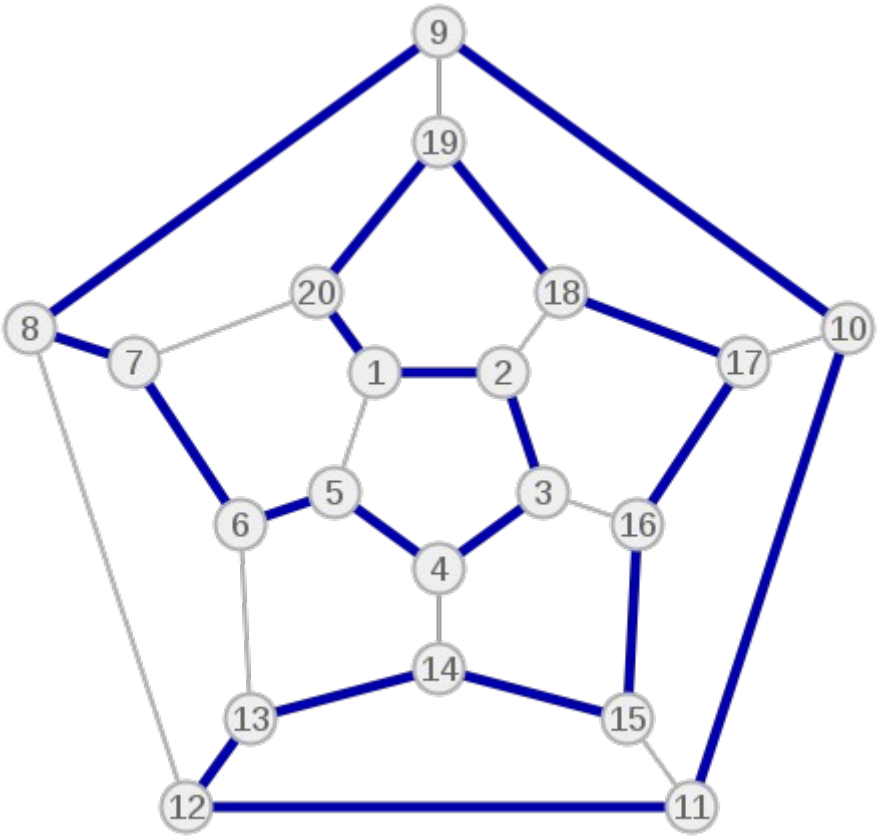
Пример
$(\{a,b\}, \{a^n b^n\} \mid n \in N_0$

Вход: $x \in \{a,b\}^*$

Выход : Да, если $x=a^n b^n$ для некоторого $n \in N_0$
Нет – в противном случае.

Замечание к *проблеме существования Гамильтонова цикла (HC),*

Гамильтонов путь—это путь, содержащий каждую вершину графа ровно один раз. Гамильтонов путь, начальная и конечная вершины которого совпадают, называется **гамильтоновым циклом**. Гамильтонов цикл является простым остовным циклом. Задача определения содержит ли данный граф гамильтонов цикл является NP-полной. К этой теме вернемся при изучении современных алгоритмов поиска для систем искусственного интеллекта.



Замечание к анализу алгоритмов (практика 1-2-3-4)

Задача имеет алгоритм, который правильно ее решает. Насколько это решение эффективно?

Последовательность действий алгоритма определяется не в последнюю очередь входными данными.

Вы ищите максимальный элемент в списке.

```
maximum=list[1]
for i=2 to N do
  if(list[i]>maximum)
     then  maximum=list[i]
  end if
end for
```

Если ваш список упорядочен в порядке убывания, то перед началом цикла будет сделано одно присваивание, а в теле цикла присваиваний вообще не будет.

Если список упорядочен по возрастанию, то всего будет сделано N присваиваний(одно перед началом цикла и N-1 внутри цикла)

Это знаит, что вид водного множества приводит к разной скорости решения задачи. Это значит, что надо рассматривать все типы входных множеств.

Поэтому мы говорит о классах входных данных.
Множество из 10 различных чисел может быть размещено в списке 3 628 800 способами. Поэтому есть смысл разбить списки на классы в зависимости от того, что делает наш алгоритм.
В нашей задачи для нас представляет интерес местоположение максимума., поэтому у нас получится 10 классов. И мы можем рассмотреть поведение алгоритма на одном только множестве из каждого класса., потому что на множествах входных данных одного класса алгоритм производит одинаковое количество операций.

И в случае с сортировкой выбор входных данных существенно влияет на скорость выполнения алгоритма. Алгоритм сортировки может работать очень быстро, если входной список уже отсортирован., но другие алгоритмы на этом списке могут показать результаты гораздо хуже. А на случайном списке результат может оказаться вполне приличным. Поэтому при анализе поведения алгоритма нельзя ограничиться анализом поведения алгоритма на одном входном наборе данных.Надо попытаться найти как самый быстрый результат, так и самый медленный. Поэтому и предлагаются – наилучший случай, средний случай и наихудший случай. Это совсем непросто.

Лекция 12

Сложность по Колмогорову

Как измерить количество информации в словах?
Что такое колмогоровская сложность?

Все мы архивировали свои файлы (arj, rar, zip, и т.д.). Это происходит с помощью программ, которые сжимают эти файлы. Применив такую программу к некоторому файлу, содержащему текст или программу, мы на выходе получаем его сжатую версию. Она, как правило, короче исходного файла. По ней можно восстановить исходный файл с помощью парной программы -декомпрессора.

В первом приближении колмогоровскую сложность можно описать как длину его сжатой версии.

И тогда файл, имеющий регулярную структуру и хорошо сжимаемый, имеет малую колмогоровскую сложность (в сравнении с его длиной). Наоборот, плохо сжимаемый файл имеет сложность, близкую к длине.

Имеем двоичное слово (конечная последовательность битов)
Слово 1001 имеет длину = 4, пустое слово имеет длину=0

1 0 0 ……1 0 1

1) малая колмогоровская сложность после декомпрессии

2) большая колмогоровская сложность(близка к длине)

Мы будем рассматривать только слова над булевским алфавитом Σ_{bool} и измерять информацию следующим образом. Будем считать, что слово w содержит мало информации, если есть короткое представление этого слова (то есть если оно сжимаемое)

Слово содержит много информации, если не существует короткого представления w. (то есть такого представления, которое было бы короче, чем |w|). Интуитивно ясно, что слово с малым информационным содержанием является регулярным, и значит, его можно легко описать. Слово с высоким информационным содержанием нерегулярно (то есть является случайным распределением нулей и единиц) Поэтому такое слово приходится записывать постепенно, бит за битом.

Пример

1) 011 011 011 011 011 011 011 011 --- слово1)
2) 01011010001011010011101100 10 ---- слово 2)

Слово 1) имеет короткое представление $(011)^{8,}$, содержит меньше информации, чем слово 2).

Процесс построения некоторого представления слова w ,которое короче, чем само w, и может быть короче, чем предыдущий вариант его представления, называется *сжатием* слова w. Используя терминологию теории вычислимых функций, можно сказать, что способ описания есть вычислимая функция из $(\Sigma_{bool})^*$ в $(\Sigma_{bool})^*$, то есть во множество всех двоичных слов.

Далее выбираем и фиксируем какой-либо метод сжатия - и длину сжатого представления слова используем как меру его информационного содержания.

Сжатое представление слова должно быть словом над алфавитом Σ_{bool} , то есть нельзя использовать при сжатии более мощный алфавит, такое сжатие не будет настоящим сжатием.

Но проблема в том, что можно предложить бесконечно много различных методов сжатия - какой из них будет самым правильным?

Как получить объективную меру информационного содержания слов? И чтобы эта мера могла быть использована в различных областях информатики?

Выход из этой ситуации предложил Колмогоров. Он ввел в рассмотрение алгоритм (программу), чтобы найти способ измерения информационного содержания.

Определение: Для любого $x \in (\Sigma_{bool})^*$ *сложность по Колмогорову K(x) слова x* – это двоичная длина наиболее короткой программы на Паскале, которая генерирует x.

Двоичное слово x –> D(x) –> двоичное слово y.

D(x) – алгоритм(программа)- декомпрессор;

D(x)=y

Здесь мы рассматриваем программу восстановления (речь идет не о программе сжатия). У нас работает декомпрессор – произвольный алгоритм (программа) , который получает на вход двоичное слово x и выдает на выход тоже двоичное слово y.

X является описанием y при данном D(относительно данного D). D – это способ описания. D может быть определен не на всех словах. При некоторых x вычисление D(x) может зациклиться. Мы также не накладываем ограничения на время работы D.

Машинный код правильной программы можно рассматривать как слово над алфавитом $\Sigma_{bool}=\{0,1\}$. Следовательно, для каждого слова $x \in (\Sigma_{bool})^*$ мы рассматриваем все машинные коды программ, генерирующих x , (их бесконечно много), и длина самой короткой из этих программ объявляется сложностью по Колмогорову слова x.

$KS_D(x) = \min \{dl(y) \mid D(y)=x\}$ dl –длина слова y.

Индекс D говорит о том, что определение зависит от способа D. Путем подбора способа описания D можно для любого слова x добиться, чтобы x имел малую сложность. ($D(\lambda)=x$, и тогда $KS_D(x)=0$.

Колмогоровскую сложность слова x можно назвать также количеством информации в слове x.

Слово, состоящее из одних нулей, может быть описано очень коротко, но содержит мало информации, а сложное слово, которое не поддается сжатию, содержит много информации. Если слово x имеет сложность k , мы говорим, что x содержит k битов информации. Естественно ожидать, что число битов информации в слове не превосходит его длины, Одно из основных свойств сложности по Колмогорову гласит:

Лемма: Существует константа d такая, что для каждого $x \in (\Sigma_{bool})^*$ выполняется неравенство

$$K(x) <= |x| + d . \qquad d\text{- const}$$

Замена одного оптимального способа на другой оптимальный способ приводит к изменению сложности не более, чем на константу.

Не имеет смысла говорить о KS конкретного слова x , не указывая , какой оптимальный способ мы используем. Выбирая тот или иной способ, можно сделать любое из двух слов более простым. Свойства оптимального способа зависят от используемого языка программирования.(то есть способа записи программы).. Два таких способа приводят к сложностям, отличающимся не более, чем на константу.(которая есть длина интерпретатора одного языка программирования, написанного на другом языке.) Однако, если мы говорим о сложностях порядка миллионов (ДНК) , то уже не важно, какой именно язык мы выбрали.

Колмогоровская сложность любого слова – конечна (то есть любое слово имеет описание)

При алгоритмическом преобразовании количество информации не увеличивается. (точнее, увеличивается не более, чем на константу, а константа зависит от алгоритма преобразования.)

Это значит, что количество информации не зависит от кодирования. Если в слове заменить все нули на единицы и

наоборот, то полученное слово будет иметь ту же сложность, что и исходное.(с точностью до константы).

Определение : Сложность по Колмогорову числа $n \in N$ есть

$$K(n) = K(Bin(n)).$$

(В соответствии с определением можно формально определить информационное содержание для натуральных чисел)

Лемма: Для каждого $n \in N$ существует слово $w_n \in (\Sigma_{bool})^n$, такое что

$$K(w_n) >= | w_n | = n ,$$

То есть для любого $n \in N$ существует несжимаемое слово длины n.

(то есть существуют слова, несжимаемые относительно сложности по Колмогорову)

Несжимаемые слова очень “нерегулярны” - и поэтому они могут рассматриваться как случайные. Существует бесконечно много случайных слов над алфавитом {0,1} - по крайней мере одно для каждой длины. Сложность по Колмогорову дает возможность измерить не только объем информации, содержащийся в слове, но и уровень случайности слова.

Сложность по Колмогорову используется как инструмент для проверки полученных результатов.

Чтобы слово можно было сжать, в нем должна быть закономерность, которая и позволяет сжать слово. Поэтому естественно считать случайными несжимаемые слова, ибо случайность и есть отсутствие закономерности.

Замечание. Вопрос о количестве информации в различных объектах изучался и до алгоритмической теории информации с помощью понятия *шенноновской энтропии.*

Мера энтропии по Шеннону – это разница между информацией , содержащийся в сообщении, и той частью информации, которая точно известна. Энтропия ограничивает максимальную возможность сжатия без потерь, которая может быть реализована при использовании на практике какого – то алгоритма

кодирования, например, кодирования Хаффмана или какого-либо другого..

Если случайная величина γ принимает n значений с вероятностью p_1, p_2,….., p_n, то ее шенноновская энтропия определяется формулой

$$H(\gamma)= \sum p_i(-\log_2 p_i)$$

Говорят, что результат, имеющий вероятность p_i , несет в себе ($-\log_2 p_i$) битов информации. Тогда $H(\gamma)$ можно интерпретировать как среднее количество информации в результате случайной величины. Но при оценке большого текста возникают сложности, и использование в таких ситуациях колмогоровской сложности является более предпочтительным.

Шенноновская (классическая) теория информации измеряет количество информации *в случайных величинах.*

Колмогоров предложил измерять количество информации *в конечных объектах* с помощью теории алгоритмов, определяя *сложность объекта как минимальную длину программы, порождающую этот объект.* Объект считается случайным, если его сложность близка к максимуму.

Оказалось, что это возможно, но лишь *с точностью до ограниченного слагаемого.*

Выводы:

Алфавиты, слова, языки и способы их представления - это предмет изучения теории формальных языков. Эти основные понятия используются во всех областях обработки данных. Теория формальных языков одна из классических областей информатики. А так как объекты изучения информатики - данные, информация, программы, теоремы, доказательства, вычисления и т. д. представляются словами, то теория формальных языков является основой для других фундаментальных областей теоретической информатики – теории вычислимости, теории трансляции и т. д.

Понятие сложности по Колмогорову – это мера количества содержащейся в словах информации.