

Санкт-Петербургский государственный
университет

Вероятностная игра
”Добавь букву!”

Андреев Роман
Звонарев Никита
Тарасов Артем
322 группа

8 октября 2012 г.

Описание игры

Что наша жизнь? Игра!

М. И. Чайковский

Он складывал из льдин и целые слова, но никак не мог сложить того, что ему особенно хотелось, — слово “вечность”.

Г. Х. Андерсен

Рассмотрим одну увлекательную игру со словами. В начале у Игрока есть пустая строка, а также он знает параметр игры K — количество ходов. Каждый ход состоит из двух фаз:

1. Игроку дается произвольная буква из заданного алфавита (например строчные латинские буквы от 'a' до 'z').
2. Далее Игрок решает, добавить ли ему данную букву к его текущей строке, или нет, и сообщает результат. После этого ход заканчивается.

Есть две разновидности игры:

1. Можно добавлять буквы только в конец слова.
2. Можно добавлять буквы либо в начало, либо в конец слова.

В конце игры производится подсчет очков следующим образом:

- Если получившееся слово есть в данном Игроку заранее Словаре (например мы рассматривали стандартный английский словарь, состоящий из 581111 слов), то он получает столько очков, какова длина слова. В противном случае он не получает ничего, то есть ноль очков.

Игрок хочет максимизировать свой выигрыш.

Типовые решения

Начнем с самого основного решения.

1 Оптимальное решение

Чего нам хочется добиться к концу игры? Максимизировать значение математического ожидания длины слова, которое получает наш алгоритм. Давайте предподсчитаем для всех слов и всех K следующую величину:

$f(s, K)$:= наибольшее значение мат. ожидания результата, если нам осталось сделать K ходов, и мы уже себе набрали слово s

Заметим, что если в какой-то момент у нас получилось слово, которое не является:

1. для первого варианта игры префиксом ни одного из наших слов,
2. для второго варианта игры подстрокой ни одного из наших слов,

то для любого K значение $f(s, K)$ равно 0.

В дальнейшем будем рассматривать только строчки обладающие выше-описанным свойством.

Также заметим, что $f(s, 0) = \begin{cases} 0, & \text{если } s \text{ не из словаря} \\ \text{len}(s), & \text{если } s \text{ входит в словарь} \end{cases}$.

Теперь понятно, как подсчитать оставшиеся значения: мы перебираем, какая буква к нам придет следующей (все буквы приходят с вероятностью $\frac{1}{|Alp|}$) и если в том случае, когда мы добавляем букву, мат. ожидание больше, то мы будем ее приписывать, иначе оставим текущее слово.

Если мы будем подсчитывать величины f в порядке возрастания K (сначала у нас определены все значения с $K = 0$, а чтобы вычислить $f(s, K)$ мы будем использовать $f(s', K - 1)$, которые мы уже будем знать), то все будет посчитано корректно.

Если объединить все сказанное выше, то получаем такую формулу:

1. В первом варианте игры:

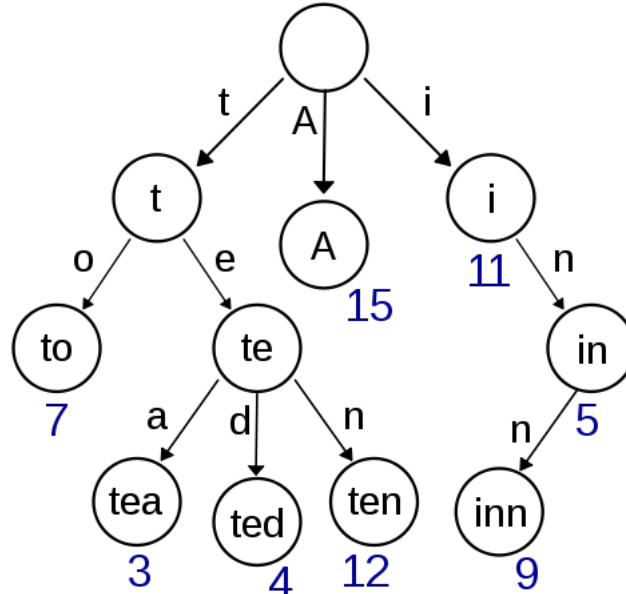
$$f(s, K) = \begin{cases} \sum_{c \in Alp} \frac{\max(f(s, K-1), f(s+c, K-1))}{|Alp|}, & \text{если } K > 0 \\ 0, & \text{если } s \text{ не является префиксом ни одной из строк из словаря} \\ \text{len}(s), & \text{если } K = 0 \text{ и } s \text{ входит в словарь} \end{cases}$$

2. Во втором варианте игры:

$$f(s, K) = \begin{cases} \sum_{c \in Alp} \frac{\max(f(s, K-1), f(s+c, K-1), f(c+s, K-1))}{|Alp|}, & \text{если } K > 0 \\ 0, & \text{если } s \text{ не является подстрокой ни одной из строк из словаря} \\ \text{len}(s), & \text{если } K = 0 \text{ и } s \text{ входит в словарь} \end{cases}$$

Теперь оценим асимптотику:

- В первом варианте: При правильном выборе структуры данных для хранения строк (в нашем случае оптимальной структурой данных является бор всех строк, в зарубежной литературе данная структура называется Trie) мы получаем предподсчет за $O\left(\sum_{s \in Dict} \text{len}(s) \cdot K\right)$ времени работы и столько же памяти.



- Во втором варианте:

Здесь все аналогично прошлому случаю, только в боре мы храним суффиксы, и получаем по сути суффиксное дерево. Тут нам также понадобятся суффиксные ссылки в боре, чтобы делать переход по символу в начале (аналогичная структура используется в алгоритмах Ахо-Корасик и Укконена). В итоге реализация потребует также $O\left(\sum_{s \in Dict} \text{len}(s) \cdot K\right)$ времени и памяти.

Заметим, что на один ход мы тратим $O(|Alp|)$ времени.

2 Жадное решение

Пусть $n = \max_{s \in Dict} \text{len}(s)$. Понятно, что когда у нас $K \gg n$, и мы будем пытаться набирать самое длинное слово, то вероятность того, что мы в итоге не сможем набрать все это слово, крайне мала. Если точнее, то K должно быть больше $n \cdot |Alp|$, ведь слово длины n мы будем набирать в среднем за $n \cdot |Alp|$ ходов, так как на одну букву в среднем будет уходить $|Alp|$ ходов.

Тогда $f(n, K) = \frac{f(n-1, K-1)}{|Alp|} + \frac{f(n, K-1) \cdot (|Alp|-1)}{|Alp|}$, где $f(0, K) = n$, если $K \geq 0$

Заметим, что мы получили события Бернулли с вероятностью успеха $\frac{1}{|Alp|}$, и нам нужно подсчитать вероятность, при которой случится больше либо равно n успешных исходов. Эта вероятность равна

$$P(K, n) = \sum_{k=n}^K \binom{K}{k} \cdot \left(\frac{1}{|Alp|}\right)^k \cdot \left(\frac{(|Alp|-1)}{|Alp|}\right)^{K-k} =$$

$$1 - \sum_{k=0}^{n-1} \binom{K}{k} \cdot \left(\frac{1}{|Alp|}\right)^k \cdot \left(\frac{(|Alp|-1)}{|Alp|}\right)^{K-k}$$

В данном случае применима интегральная теорема Лапласа-Муавра:

$$P(K, n) \approx \frac{1}{\sqrt{2\pi}} \cdot \int_{\frac{n - \frac{K}{|Alp|}}{\sqrt{K \cdot \frac{|Alp|-1}{|Alp|^2}}}}^{\infty} e^{-\frac{x^2}{2}} dx = 1 - \Phi\left(\frac{n - \frac{K}{|Alp|}}{\sqrt{K \cdot \frac{|Alp|-1}{|Alp|^2}}}\right),$$

где $\Phi(x)$ — функция распределения стандартного нормального распределения.

То есть $P(n \cdot |Alp|, n) \approx \frac{1}{2}$.

Например при $K = 1000$ и $n = 22$ (в нашем словаре именно так) получаем, что $P = 0.99868$, а мат. ожидание равно 21.9709. Оптимальное решение при $K = 1000$ получает результат 21.9855. Теперь можно утверждать, что при больших K данная эвристика близка к оптимальному решению, ведь

$$\lim_{K \rightarrow \infty} P(K, n) = 1,$$

а лучше самого длинного слова ничего набрать заведомо не получится.

Данные рассуждения допустимы только при первом варианте игры. Для игр второго типа потребуется более сложный и тщательный расчет вероятности. Нужно учитывать, что на ответ будет влиять само слово, если быть точнее, то его суффиксное дерево. Например, можно оценить снизу вероятность уже полученной ранее оценкой.

3 Улучшенное жадное решение

Рассмотрим более простую задачу, решение которой приведет к еще одной эвристике. Пусть мы хотим брать не все слова, а только какое-то подмножество словаря ($G \subset Dict$). Теперь наша задача — минимизировать среднее число ходов, которое понадобится нам, чтобы собрать слово из G . Что делать в этом случае?

Обозначим за $g(s)$ мат. ожидание количества ходов, которое нам понадобится, чтобы набрать слово из G , если уже набрали слово s .

Как ее вычислить? Если $s \in G$, то ответ 0. Если s не является префиксом ни одной из строк из G , то ответ $+\infty$. Все остальные значения будем вычислять по мере убывания длины.

Пусть мы хотим сосчитать $g(s)$. Тогда посмотрим на множество мат. ожиданий, которое мы получим, если добавим один символ:

$$A_c = g(s + c).$$

Так мы получим $|Alp|$ значений. Теперь нам нужно выбрать, в какие мы будем ходить, а в какие нет. Понятно, что ходить надо по символам с наименьшими A_c . Упорядочим эти значения по возрастанию. Сколько нам понадобится действий, если мы будем выбирать первые k значений?

$$P(k) + \frac{\sum_{i=1}^k g_i}{k},$$

где $P(k)$ — мат. ожидание количества ходов, которое потребуется, чтобы выпала одна из этих k букв.

$$P(k) = \sum_{i=1}^{+\infty} i \cdot \left(\frac{k}{|Alp|} \right) \cdot \left(\frac{|Alp| - k}{|Alp|} \right)^{i-1} = \frac{|Alp|}{k}$$

Теперь выбираем среди всех k такое, чтобы получившееся мат. ожидание было минимально. Таким образом, мы подсчитали для каждой строчки s следующее значение: если у нас сейчас набрана строчка s и нам выпал символ c надо ли нам ходить по нему, или ждать. Совершенно аналогичным способом данный метод обобщается для второго типа игры.

Получаем для обоих случаев трудоемкость $O\left(\sum_{s \in G} len(s) \cdot |Alp| \cdot \log |Alp|\right)$ по времени и $O\left(\sum_{s \in G} len(s)\right)$ по памяти. Но если мы будем хранить только

необходимые переходы, то мы получим $O\left(\sum_{s \in G} len(s) \cdot \log |Alp|\right)$ времени.

Например, если $G = \{\text{множество слов длины 10 из нашего словаря}\}$, то по этой стратегии мы будем приходить в них примерно за 160 ходов, что сильно лучше, чем $|Alp| \cdot 10 = 26 \cdot 10 = 260$.

Данная структура дает широкий простор для эвристик. Например, мы можем положить G равным $\{\text{множество слов из словаря длины } \geq L\}$ или $\{\text{множество слов из словаря длины ровно } L\}$ для какого-то фиксированного L . Далее возникает вопрос, какое L выбрать. Так как в нашем словаре максимальная длина слова небольшая, то можно просто перебрать все L .

Итог

Что же мы получили в итоге? У нас есть 3 решения для различных значений K . Когда K небольшое и нам хватает памяти, то мы запускаем

оптимальное решение. Когда K уже достаточно большое, то мы запускаем жадное решение, которое пытается набрать самое длинное слово, причем памяти ему нужно константное количество. В остальных случаях мы пробуем разные эвристики из последнего пункта и выбираем ту, которая работает лучше всего. На нашем словаре второй случай можно не реализовывать, так как памяти оптимальному решению хватает и при $K = 1000$ и оно успевает быстро отработать.