Задача о наименьшей надстроке

Торунова Анастасия.394 группа

19 декабря 2015 г.

Формулировка задачи.

Дано множество слов $A = \{\alpha_1, \alpha_2 \ldots, \alpha_n\}$ над конечным алфавитом и число k. Нужно найти такую строку w, что все строки из множества A являются подстроками w и длина $w \leq k$. Заметим, что можно считать, что $k < \sum_{k=1}^{n} |\alpha_i|$, так как иначе можно всегда взять конкатенацию α_i-ых.

NP - полнота.

Утверждение. *Задача о наименьшей надстроке* **NP***-полна.*

Доказательство. Сначала покажем,что эта задача лежит в **NP**. Воспользуемся сертификатным определением. Тогда сертификатом будет сама надстрока w, а верификатор проверяет, что $|w| \leq k$ и если это не выполняется, то выдает 0, а если выполняется то продолжает проверку того, что $\forall \alpha_i \in A$ α_i – подстрока w. Поскольку проверка на подстроку выполняется за $O(|\alpha_i| * |w|)$ при наивном алгоритме, а проверка на длину за $O(k)$, то верификатор будет всегда работать полиномиальное время от длины входа задачи. Таким образом, задача лежит в **NP**. Теперь покажем, что она **NP**-трудная. Для этого воспользуемся **NP**-полнотой задачи $\mathsf{DHAMPATH} = \{(G, s, t)|$ в орграфе G есть гамильтонов путь из вершины s в вершину t $\}$. Пусть у нас есть вход для задачи $\mathsf{DHAMPATH}$: граф G и вершины s и t. Пусть $V = v_1, \ldots v_n$ – множество вершин графа G, где $v_1 = s$, $v_n = t$, а $E = \{e_1, \ldots e_m\}$ – множество его ребер. Рассмотрим алфавит $\Sigma = V \cup V' \cup \{@, \#, \$\}$, где $V' = \{\bar{v}|v \in V \setminus \{v_n\}\}$. Пусть $OUT(v)$ - множество вершин, достижимых из v по ребру. Тогда определим следующие множества строк:

$A_v = \{\bar{v}u\bar{v}|u \in OUT(v)\} \cup \{u_i\bar{v}u_{i+1}|u_i \in OUT(v)\}$ $\forall v \in V \setminus \{v_n\}$ Здесь и везде далее под сложением $i + 1$ подразумевается сложение по модулю $|OUT(v)|$.

$C = \{v\#\bar{v}|\forall v \in V \setminus \{v_1, v_n\}\}$

$T = \{@\#\bar{v_1}, \bar{v_n}\#\$\}$

$$L = C \cup T \cup \bigcup_{v \in V \setminus \{v_n\}} A_v$$

Тогда покажем, что в G есть гамильтонов цикл $\Leftrightarrow$ для L есть надстрока размера $\leq 2m+3n$.

Пусть в G есть гамильтонов путь $v_1 = w_1, w_2, \dots w_n = v_n$. Пусть для вершины v ребро (v, u_i) лежит на этом пути. Тогда построим надстроку для A_v следующим образом: $\bar{v}u_i\bar{v}u_{i+1}\bar{v}\dots\bar{v}u_{i+|OUT(v)|}\bar{v}u_i$. Назовем эту строку $S(v, u_i)$. Тогда для нашего гамильтонова пути построим строку пересекая последовательно следующие строки:

$$@\#\bar{v_1}, S(w_1, w_2), \bar{w_2}\#w_2, S(w_2, w_3)\dots, S(w_{n-1}, w_n), \bar{v_n}\#\$$$

Эта строка является надстрокой всего множества L, так как содержит надстроки для каждого A_v и все строки из C и T.

Она имеет длину $\sum_{v\in V\setminus\{v_n\}} 2*(|OUT(v)|+1)+(n-2)+4 = 2m+2(n-1)+n-2+4 = 2m+3n$ Таким образом мы построили нужную надстроку. Теперь пусть у нас есть надстрока длины $\leq 2m+3n$ для множества L. Покажем, что тогда в G есть гамильтонов путь. Сначала заметим, что $|L| = 2m+n$. Тогда их общая длина $3(2m+n)$, так как длина каждой строчки 3. Посмотрим, насколько маленькой можно сделать надстроку этих строк. Все строки в L могут пересекаться не более чем по 2 символа с каждой стороны,так как они не могут совпадать. В случае, когда они все пересекаются по 2, мы получаем надстроку длины $3(2m+n)-2(2m+n-1) = 2m+n+2$. Заметим, что строки из множества T могут пересекаться только по одному символу с остальными строками из L, а строки из C могут пересекаться только по 2 символа - по одному с каждой стороны, так как у них в центре #. А строчки из множеств A_v допускают пересечение по 2 символа с каждой стороны. Тогда минимальная длина надстроки получается равной $2m+n+2+2(n-2)+2 = 2m+3n$. А так как длина нашей надстроки $\leq 2m+3n$, то она в точности $2m+3n$. Заметим, что тогда у нашей надстроки первая и последняя подстроки лежат в T, так как если бы строки из T были бы внутри надстроки, то для каких-то двух строк из оставшихся пересечение с ними было бы 0 с одной из сторон, так как строки из T допускают пересечение только с одной стороны. Что увеличивало бы длину строки хотя бы на 2 по сравнению со случаем, когда строки из T расположены по краям. Но у нас длина строки минимально возможная, поэтому строки из T расположены по краям надстроки. Теперь рассмотрим участок нашей надстроки между двумя последовательными #. Тогда он начинается с $\bar{v_i}$ и заканчивается на v_j. Поскольку внутри этого участка нет #, то он состоит из пересекающихся строчек из A_v для некоторых v. Поскольку наша надстрока минимального размера, то как было показано выше, эти строчки должны пересекаться по 2 символа с каждой стороны, кроме строк по краям участка. С другой стороны каждая строчка из A_v должна пересекаться с соседями по 2 символам с каждой стороны. И для каждой строчки из A_v $\exists!$ строки, пересекающиеся с ней по 2-м символам, причем они тоже лежат в A_v. Тогда получаем, что участок между двумя # должен содержать все строчки из A_v для некоторого v. Применяя все сказанное выше к каждой паре последовательных #, получаем, что наша строка имеет вид последовательно пересекающихся строк в следующем порядке:

$$@\#\bar{v_1}, S(w_1, w_2), \bar{w_2}\#w_2, S(w_2, w_3)\dots, S(w_{n-1}, w_n), \bar{v_n}\#\$$$

Где $v_1 = w_1, w_n = v_n$. Но тогда по построению строк, получаем, что $v_1 = w_1, w_2, \dots w_n = v_n$ - гамильтонов путь в G из $v_1 = s$ в $v_n = t$. Что и требовалось. Осталось показать, что построение множества строк происходит за полиномиальное время. Построение множеств C и T происходит за $O(|V|)$. А построение всех множеств A_v за $O(|V|^2)$ В итоге получаем полиномиальное построение L. Значит, мы построили полиномиальное сведение задачи DHAMPATH к нашей задаче. Значит наша задача является **NP**-трудной. А значит и **NP**-полной. □

Алгоритм, дающий 4-приближение.

Рассмотрим строки α_i и α_j. Пусть v - строка максимальной длины такая, что $\alpha_i = uv$, $\alpha_j = vw$, где $|u| > 0$ и $|w| > 0$. Тогда определим функцию $overlap(\alpha_i, \alpha_j) = v \ \forall i, j \in \{1 \dots n\}$. Аналогично, определим функцию $pref(\alpha_i, \alpha_j) = u$.

Пусть мы нашли кратчайшую надстроку, тогда порядок, в котором строки из исходного набора присутствуют в ней, назовем оптимальным. Пусть оптимальная длина строки это OPT, тогда:

$$\mathsf{OPT} = \sum_{i=1}^{n} |pref(s_i, s_{i+1})| + |overlap(s_n, s_1)|$$

где $s_1 \dots s_n$ – оптимальный набор, а сложение в индексах идет по модулю n. Это так, поскольку строки образуют надстроку, пересекаясь своими *overlap*ами. Последнее слагаемое в этом выражении появляется, чтобы добавить последнюю подстрочку в надстроку.

Теперь рассмотрим полный ориентированный граф G, в котором вершинами будут исходные строки, а вес ребра из $i-$ой строки в j-тую будет равен $|pref(\alpha_i, \alpha_j)|$. Заметим, что если взять какой-то гамильтонов цикл в этом графе (допустим по вершинам $i_1 \dots i_n$), то рассматривая строку, получаемую последовательным написанием строк $\alpha_{i_1} \dots \alpha_{i_n}$ со склеиванием *overlap*ов, получаем одну из надстрок исходного набора. Обозначим длину надстроки, получаемой из решения задачи коммивояжера на данном графе, как TSP. Тогда понятно, что $TSP \leq OPT$. Но поскольку мы не умеем решать TSP за полиномиальное время, воспользуемся задачей о покрытии графа циклами, не пересекающимися по вершинам, с минимальным суммарным весом всех циклов.

Сначала поймем, как это помогает решить исходную задачу. Строка, сформированная из каждого цикла тем же способом, что и раньше, образует его надстроку. Поскольку циклы не пересекаются по вершинам и покрывают весь граф, то для каждой строки исходного набора есть надстрока, в которой она присутствует. Тогда конкатенацией всех полученных надстрок мы получим надстроку для всего набора.

Теперь обозначим длину надстроки, получающейся таким способом, как MCC(Miniimum Cycle Cover). Заметим, что гамильтонов цикл в частности является покрытием циклами нашего графа, поэтому решение $MCC \leq TSP$. А значит, мы имеем неравенство: $MCC \leq TSP \leq OPT$.

Теперь рассмотрим алгоритм решения задачи о циклах. Построим по нашему графу G двудольный граф G' следующим образом: если в исходном графе есть ребро (α_i, α_j), то мы проводим ребро (v_i, u_j) в G' с таким же весом, как в исходном графе, где $\{u_1 \dots u_n\}$ - левая доля G', а $\{v_1 \dots v_n\}$ - правая. Тогда нахождение покрытия циклами сводится к нахождению совершенного паросочетания минимального веса. Действительно, если в G' есть совершенное паросочетание, то выйдем из некоторой вершины u_i и начнем ходить по ребрам паросочетания следующим образом: если мы перешли в вершину v_j правой доли, то если вершина с соответствующим индексом из левой доли не посещена, телепортируемся в нее и продолжаем идти. Если соответствующая вершина посещена, то ищем в левой доле непосещенную вершину и идем в нее. Если непосещенных вершин левой доли не осталось, то завершаемся. Таким обходом мы обойдем все вершины левой доли, а значит и правой, причем по одному разу. Поэтому если согласно построению графа G' перенести все ребра этого обхода на граф G, то получится его покрытие циклами, так как тогда мы будем идти последовательно из вершины α_i по смежным вершинам в G без повторов, до тех пор пока не дойдем опять до α_i. После этого цикл замкнется, мы выберем другую(еще не посещенную) вершину и продолжим обход из нее.Таким образом, мы показали, что из совершенного паросочетания получается покрытие циклами. А поскольку веса ребер в G' были равны весам соответствующих ребер в G, то при нахождении совершенного паросочетания минимального веса мы автоматически находим минимальное покрытие циклами. Остается заметить, что для паросочетания минимального веса существует полиномиальный алгоритм (за $O(n^3)$).

Теперь остается сделать конкатенацию строк из полученных циклов. Пусть $C_1 \dots C_n$ - полученные циклы. Тогда веса каждого из них это $w(C_i) = \sum_{j=1}^{n} |pref(\alpha_{i_j}, \alpha_{i_{j+1}})|$ по построению графа. Пусть строка, полученная из цикла это $s(C_i)$. Тогда ее длина это $|s(C_i)| = w(C_i) + |l_i|$, где l_i – последняя подстрока цикла. Тогда получаем, что длина полученной надстроки это $SUPERSTRING = \sum_{i=1}^{n} |s(C_i)| = \sum_{i=1}^{n} w(C_i) + \sum_{i=1}^{n} |l_i|$. Мы уже знаем из предыдущих рассуждений, что $\sum_{i=1}^{n} w(C_i) < OPT$. Осталось показать, что $\sum_{i=1}^{n} |l_i| < 3 * OPT$ Для этого докажем следующее утверждение.

Утверждение. *Пусть C_1 и C_2 – циклы из минимального покрытия, строки $s_1 \in C_1$, $s_2 \in C_2$. Тогда $|overlap(s_1, s_2)| < w(C_1) + w(C_2)$.*

Доказательство. Предположим, что $|overlap(s_1, s_2)| \geq w(C_1) + w(C_2)$. Выберем в качестве $s(C_1)$ и $S(C_2)$ строки, начинающиеся с s_1 и s_2 соответственно. Пусть $\forall s$ s^∞ – конкатенация $sss\dots$. Пусть $x = overlap(s_1, s_2)$, тогда поскольку $s_1 \in s(C_1)^\infty$, то $x \in s(C_1)^\infty$, аналогично, $x \in s(C_2)^\infty$.

Пусть x_1 – префикс x длины $w(C_1)$, а x_2 – префикс x длины $w(C_2)$ (мы можем выделить такие префиксы, так как по предположению $|x| > w(C_1)$ и $|x| > w(C_2)$). Тогда получаем, что на цикле C_1 написана строчка x_1, а на цикле C_2 написана строчка x_2 (если пройти по циклу и склеить соседние строчки по их пересечению). Тогда так как $x \in C_1$, то x – префикс x_1^∞, аналогично x – префикс x_2^∞. Поскольку из предположения $|x| > |x_1| + |x_2|$, то получаем, что x_1x_2 – префикс x и что одновременно x_2x_1 – префикс x. Но так как их длины совпадают, то $x_1x_2 = x_2x_1$. Докажем по индукции, что $\forall k\ x_1^kx_2^k = x_2^kx_1^k$. База при $k = 1$ уже доказана. Пусть теперь выполнено, что $x_1^{k-1}x_2^{k-1} = x_2^{k-1}x_1^{k-1}$. Тогда $x_1^kx_2^k = x_1x_2^{k-1}x_1^{k-1}x_2$. Теперь пользуясь базой, переставим x_1 и x_2^{k-1} и аналогично x_2 и x_1^{k-1}. Тогда получаем, что $x_1^kx_2^k = x_2^{k-1}x_1x_2x_1^{k-1} = x_2^kx_1^k$. Переход доказан. Итого получили, что $\forall k\ x_1^kx_2^k = x_2^kx_1^k$, но тогда из этого следует, что $x_1^\infty = x_2^\infty$. Но тогда получаем, что так как x_1 – строка, написанная на цикле C_1, а x_2 – строка, написанная на цикле C_2, то все строки лежащие на обоих циклах – подстроки x_1^∞. А значит, поскольку граф подстрок полный, то там есть цикл, соединяющий все эти строки веса $w(C_1)$, а значит вместо двух циклов мы могли бы взять в покрытие только один и его вес бы уменьшился, что противоречит минимальности покрытия. Пришли к противоречию, значит в действительности $|overlap(s_1, s_2)| < w(C_1) + w(C_2)$. □

Теперь, используя доказанное утверждение, получаем, что $\forall i, j\ overlap(l_i, l_j) < w(C_i) + w(C_j)$. Поэтому если без ограничения общности считать, что l_i пронумерованы в порядке вхождения в оптимальную строку, мы получим $OPT > \sum_{i=1}^n |l_i| - \sum_{i=1}^n overlap(l_i, l_{i+1}) > \sum_{i=1}^n |l_i| - 2 * \sum_{i=1}^n w(C_i)$ А тогда имеем:

$$\sum_{i=1}^n |l_i| = \sum_{i=1}^n |l_i| - 2 * \sum_{i=1}^n w(C_i) + 2 * \sum_{i=1}^n w(C_i) < OPT + 2 * OPT = 3 * OPT$$

Получили, что нужно. Значит наш алгоритм ошибается не более, чем в 4 раза.

Итоговый алгоритм:

1.Строим граф G по набору строк как описано выше.

2.В нем находим покрытие циклами минимального веса(используя граф с совершенным паросочетанием минимального веса)

3.Для каждого цикла составляем надстроку.

4.Делаем конкатенацию всех полученных надстрок.

Исследование работы жадного алгоритма.

Известно, но не доказано, что жадный алгоритм, который на каждом шаге выбирает строки с наибольшей длиной *overlap*а и их склеивает, дает 2-приближение. Ниже приведена реализация этого алгоритма на Python.

```python
def overlap(str1, str2):
```

```
    for i in range(0, len(str1)):
        if str1[i:] == str2[0:(len(str1) - i)]:
            return str1[i:]
    return ""

def merge(str1, str2):
    overLap = overlap(str1, str2)
    return str1+str2[len(overLap):]

def greedy(strings):
    while (len(strings) > 1):
        maxOverlap,maxI,maxJ,currOverlap= 0,0,1,0
        for i in range(0,len(strings)):
            for j in range(0,len(strings)):
                if i != j:
                    currOverlap = len(overlap(strings[i], strings[j]))
                    if currOverlap > maxOverlap:
                        maxI,maxJ,maxOverlap = i,j, currOverlap
        strings[maxI] = merge(strings[maxI],strings[maxJ])
        strings.pop(maxJ)
    return strings[0]
```

Проверка на 2-приближение проводилась следующим образом: генерировалась строка, затем она разбивалась на подстрочки и для полученного набора подстрок запускался жадный алгоритм. Затем длина строки, выданной жадным алгоритмом сравнивалась с длиной исходной строки. Код тестирующих функций приведен ниже. При этом разбиение на подстрочки происходило двумя способами: в первом случае исключались подстроки, имеющие в качестве подстрок или надстрок уже выбранные ранее строки (попытка уменьшить количество вложенных подстрок), во втором случае брались произвольные подстроки. Код тестирующих функций приведен ниже.

```
def generate_string(length):
    str=""
    for i in range(0,length):
        str += random.choice("AGTC")#string.ascii_lowercase
    return str

def break_string(Str,substrFree=True):
    strings,intervals=[],[]
    for i in range(0,2*len(Str)):
        left = random.randint(0,len(Str)-2)
        right = random.randint(left+1,len(Str))
        inInterval = False
        for interval in intervals :
            if interval[0] <= left and interval[1] >= right :
                inInterval = True
```

```
            if interval[0] >= left and interval[1] <= right:
                inInterval = True
        if (not substrFree or not inInterval) :
            intervals.append([left,right])
            strings.append(Str[left:right])
    return list(set(strings))


def test(numberOfTests, length):
    resultFile = open("results.txt","w")
    maxApproxRatio = 0
    random.seed()
    for i in range(0,numberOfTests):
        testString = generate_string(length)
        substrings = break_string(testString,True)
        resultFile.write(str(i) + ". " + testString + "\n")
        resultFile.write(str(substrings) + "\n")
        greedyString = greedy(substrings)
        approxRatio=float(len(greedyString))/len(testString)
        if maxApproxRatio < approxRatio:
            maxApproxRatio = approxRatio
        resultFile.write("greedy output: " + greedyString +
            " " + str(approxRatio) + "\n")
    resultFile.write("maxRatio: " + str(maxApproxRatio))
```

На тестах с английским алфавитом получились следующие результаты: В первом случае из-за малого (существенно меньше длины исходной строки) количества подстрок в наборе, даже при большой длине исходной строки, максимальное отношение длин не превышало 2. Во втором случае, при длине исходной строки ≤ 6 действительно наблюдается максимальное отношение длин 2. На алфавите состоящем из букв G,C,T,A (первые буквы в названиях нуклеотидов) результаты такие же, как и на английском алфавите. В итоге получилось, что при длине строк ≤ 6 жадный алгоритм действительно ошибается не более, чем в два раза.

Список литературы

[1] Michael R. Garey, David S. Johnson *"Computers and Intractability. A Guide to the theory of NP-completeness"*.

[2] John Gallant, David Maier, James A. Storer *"On finding minimal length superstrings"*

[3] Avrim Blum, Tao Jiang, Ming Li, John Tromp, Mihalis Yannakakis *"Linear Approximation of Shortest Superstrings"*