

Оглавление

0.1	Первая часть	4
0.1.1	Вопрос 1	4
0.1.2	Вопрос 2	5
0.1.3	Вопрос 3	5
0.1.4	Вопрос 4	5
0.1.5	Вопрос 5	6
0.1.6	Вопрос 6	7
0.1.7	Вопрос 7	7
0.1.8	Вопрос 8	8
0.1.9	Вопрос 9	8
0.1.10	Вопрос 10	9
0.1.11	Вопрос 11	10
0.1.12	Вопрос 12	11
0.1.13	Вопрос 13	12
0.1.14	Вопрос 14	13
0.1.15	Вопрос 15	13
0.1.16	Вопрос 16	13
0.1.17	Вопрос 17	14
0.1.18	Вопрос 18	14
0.1.19	Вопрос 19	14
0.1.20	Вопрос 20	16
0.1.21	Вопрос 21	16
0.1.22	Вопрос 22	17
0.1.23	Вопрос 23.	17
0.1.24	Вопрос 24	18
0.1.25	Вопрос 25	18
0.1.26	Вопрос 26	20
0.1.27	Вопрос 27	20
0.1.28	Вопрос 28	21
0.1.29	Вопрос 29	22
0.1.30	Вопрос 30	23
0.1.31	Вопрос 31	23
0.1.32	Вопрос 32	23
0.1.33	Вопрос 33	24
0.1.34	Вопрос 34	25
0.1.35	Вопрос 35	25
0.1.36	Вопрос 36	26
0.1.37	Вопрос 37	27
0.1.38	Вопрос 38	27

0.1.39	Вопрос 39	28
0.1.40	Вопрос 40	28
0.1.41	Вопрос 41	28
0.1.42	Вопрос 42	29
0.1.43	Вопрос 43	29
0.1.44	Вопрос 44	30
0.1.45	Вопрос 45	30
0.1.46	Вопрос 46	31
0.1.47	Вопрос 47	32
0.1.48	Вопрос 48	32
0.1.49	Вопрос 49	33
0.1.50	Вопрос 50	34
0.1.51	Вопрос 51	34
0.1.52	Вопрос 52	35
0.1.53	Вопрос 53	35
0.1.54	Вопрос 54	35
0.1.55	Вопрос 55	35
0.1.56	Вопрос 56	36
0.1.57	Вопрос 57	36
0.1.58	Вопрос 58	36
0.1.59	Вопрос 59	37
0.1.60	Вопрос 60	38
0.1.61	Вопрос 61	38
0.1.62	Вопрос 62	39
0.1.63	Вопрос 63	39
0.1.64	Вопрос 64	40
0.1.65	Вопрос 65	41
0.1.66	Вопрос 66	41
0.1.67	Вопрос 67	41
0.2	Вторая часть	42
0.2.1	Вопрос 1	42
0.2.2	Вопрос 2	43
0.2.3	Вопрос 3	44
0.2.4	Вопрос 4	45
0.2.5	Вопрос 5	47
0.2.6	Вопрос 6	47
0.2.7	Вопрос 7	48
0.2.8	Вопрос 8	48
0.2.9	Вопрос 9	49
0.2.10	Вопрос 10	49
0.2.11	Вопрос 11	50
0.2.12	Вопрос 12	51
0.2.13	Вопрос 13	52
0.2.14	Вопрос 14	52
0.2.15	Вопрос 15	52
0.2.16	Вопрос 16	53
0.2.17	Вопрос 17	53
0.2.18	Вопрос 18	54

0.2.19 Вопрос 19	54
0.2.20 Вопрос 20	55
0.2.21 Вопрос 21	56
0.2.22 Вопрос 22	56
0.2.23 Вопрос 23	57
0.2.24 Вопрос 24	57

Первая часть

Вопрос 1

Двудольный граф — граф, множество вершин которого можно разбить на две части таким образом, что каждое ребро графа соединяет какую – то вершину из одной части с какой-то вершиной другой части, то есть не существует ребра, соединяющего две вершины из одной и той же части.

Двудольный граф — граф, вершины которого можно разбить на два множества так, чтобы не существовало ребёр, соединяющих вершины одного множества.

Эквивалентное определение **двудольного графа** — граф G , хроматическое число которого $\chi(G) \leq 2$.

Двудольный граф с n вершинами в одной доле и m во второй обозначается $K_{n,m}$.

Паросочетание M в двудольном графе — подмножество рёбер двудольного графа, никакие два ребра которого не имеют общей вершины.

Наибольшее паросочетание (по включению) — паросочетание M в графе G , которое не содержится ни в каком другом паросочетании этого графа, то есть к нему невозможно добавить ни одно ребро, которое бы являлось не смежным ко всем рёбрам паросочетания.

Цепью длины k назовём некоторый простой путь (т.е. не содержащий повторяющихся вершин или рёбер), содержащий k рёбер.

Чередующаяся цепь (в двудольном графе, относительно некоторого паросочетания) — цепь, в которой рёбра поочередно принадлежат/не принадлежат паросочетанию.

Увеличивающаяся цепь (в двудольном графе, относительно некоторого паросочетания) — чередующуюся цепь, у которой начальная и конечная вершины не являются концами рёбер паросочетания.

Чередование вдоль увеличивающейся цепи

По определению первое ребро увеличивающейся цепи P не принадлежит паросочетанию, второе — принадлежит, третье — снова не принадлежит, четвёртое — принадлежит, и т.д. Поменяем состояние всех рёбер вдоль цепи P : те рёбра, которые не входили в паросочетание (первое, третье и т.д. до последнего) включим в паросочетание, а рёбра, которые раньше входили в паросочетание (второе, четвёртое и т.д. до предпоследнего) — удалим из него.

Вопрос 2

Теорема Бержа.

Паросочетание M в двудольном графе G является максимальным (по мощности) тогда и только тогда, когда в G нет увеличивающей цепи относительно M .

Вопрос 3

Алгоритм Куна поиска максимального паросочетания.

Возьмём пустое паросочетание и пока удаётся найти увеличивающую относительно текущего паросочетания цепь будем выполнять чередование паросочетания вдоль этой цепи. Как только такую цепь найти не удалось, процесс останавливаем — текущее паросочетание и есть максимальное.

Асимптотика.

Алгоритм Куна можно представить как серию из n запусков обхода в глубину на всём графе. Следовательно, всего этот алгоритм исполняется за время $O(n \cdot m)$, где m — количество рёбер. Тогда в худшем случае алгоритм работает за $O(n^3)$.

Если явно задано разбиение графа на две доли размером n_1 и n_2 , то можно запускать dfs только из вершин первой доли, поэтому весь алгоритм исполняется за время $O(n_1 \cdot m)$. В худшем случае это составляет $O(n_1^2 \cdot n_2)$.

Вопрос 4

Вершинным покрытием графа $G = (V, E)$ называется такое подмножество $S \subset V$, что любое ребро этого графа инцидентно хотя бы одной вершине из S .

Минимальное вершинное покрытие — вершинное покрытие, состоящее из наименьшего числа вершин.

Алгоритм построения минимального вершинного покрытия в двудольном графе

1. Построить максимальное паросочетание (алгоритм Куна).
2. Ориентировать ребра:
 - Из паросочетания — из правой доли в левую.
 - Не из паросочетания — из левой доли в правую.
3. Запустить обход в глубину из всех свободных (не инцидентных ребрам из паросочетания) вершин левой доли, построить множества L_+, L_-, R_+, R_- (L, R — правая и левая доли соответственно, L_+, R_+ — вершины правой и левой доли, посещенные обходом, L_-, R_- — не посещенные обходом вершины).

4. В качестве результата взять $L_- \cup R_+$.

Независимым множеством вершин графа $G = (V, E)$ называется такое подмножество $S \subset V$, что $\forall u, v \in S (u, v) \notin E$.

Алгоритм построения максимального независимого множества в двудольном графе

Сделать то же самое, но взять $L_+ \cup R_-$.

Вопрос 5

Сеть — это ориентированный граф G , в котором выделены вершины $s \neq t$, s — исток, t — сток, и в котором задана функция $cap : E \rightarrow \mathbb{N}$ на ребрах.

Поток в сети — функция $f : V \times V \rightarrow \mathbb{Z}$ со следующими свойствами:

1. Ограничение потока величиной cap :

$$\forall u, v \in V |f(u, v)| \leq cap(u, v)$$

2. Свойство сохранения потока (сколько втекает, столько и вытекает)

$$\forall v \in V \setminus \{s, t\} \sum_{u|(u,v) \in E} f(u, v) = \sum_{w|(v,w) \in E} f(v, w)$$

3. Антисимметричность

$$f(u, v) = -f(v, u)$$

Остаточная сеть G_f — сеть G с модифицированной функцией $cap_f(u, v) = cap(u, v) - f(u, v)$, где f — поток в G . При этом в G_f удаляются все ребра с нулевой cap_f .

Величина потока f — число $|f| := \sum_{v \in V} f(s, v) = \sum_{u \in V} f(u, t)$.

Разрез (S, T) — разбиение вершин сети G :

1. $s \in S, t \in T$

2. $S \cap T = \emptyset$

3. $S \cup T = V$

Величина разреза $c(S, T) := \sum_{u \in S, v \in T} cap(u, v)$.

Величина потока через разрез $f(S, T) := \sum_{u \in S, v \in T} f(u, v)$.

Вопрос 6

Лемма о потоке через разрез.

$\forall (S, T)$ — разреза $|f| = f(S, T)$ (величина потока равна величине потока через разрез).

Доказательство

Обозначим $f(A, B) = \sum_{u \in A, v \in B} f(u, v)$.

$$f(S, T) = f(S, V) - f(S, S) = f(S, V) = f(S \setminus \{s\}, V) + f(s, V) = f(s, V) = |f|$$

- 1-е равенство выполняется, так как суммы не пересекаются: $f(S, V) = f(S, S) + f(S, T)$.
- 2-е равенство выполняется из-за антисимметричности: $f(S, S) = -f(S, S) = 0$.
- 3-е равенство выполняется, как и 1-е, из-за непересекающихся сумм.
- 4-е равенство выполняется из-за сохранения потока.

Лемма

Пусть (S, T) — разрез в G . Тогда $|f(S, T)| \leq c(S, T)$.

Доказательство

$$c(S, T) - |f(S, T)| = \sum_{u \in S, v \in T} cap(u, v) - \sum_{u \in S, v \in T} |f(u, v)| =$$

$\sum_{u \in S, v \in T} (cap(u, v) - |f(u, v)|) \geq 0$, из-за ограничений пропускных способностей

$$|f(u, v)| \leq cap(u, v).$$

Вопрос 7

Теорема Форда-Фалкерсона.

Следующие свойства эквивалентны:

1. f — максимальный поток в G .
2. в G_f нет путей из s в t .
3. Величина f равна величине некоторого разреза.

Доказательство

(1) \Rightarrow (2)

Пусть f — максимальный, но в G_f есть путь d из s в t . Рассмотрим $x = \min_{e \in d} (cap_f(e)) > 0$ на этом пути. Тогда вдоль этого пути можно пустить x единиц потока и увеличить f . Противоречие.

(2) \Rightarrow (3)

S — множество вершин, достижимых из s в G_f . $t \notin S$ по условию. $T = V \setminus S$. По определению (S, T) — разрез. Докажем, что $|f| = c(S, T)$:

$$c(S, T) = \sum_{u \in S} \sum_{v \in T} cap(u, v) = \sum_{u \in S} \sum_{v \in T} f(u, v) = f(S, T) = |f|$$

Переход от cap к f корректен из-за того, что в остаточной сети нет ребра из u в v .

(3) \Rightarrow (1)

$\exists(S, T) : |f| = \text{cap}(S, T)$.

При этом $\forall(S, T)$ — разрез $f(S, T) \leq \text{cap}(S, T)$ (по лемме) $\Rightarrow f = \max$.

Алгоритм Форда-Фалкерсона поиска максимального потока.

1. Изначально поток равен 0.
2. Пока в G_f есть путь из s в t :

(a) $x = \min(\text{cap}_f \text{ на этом пути}) > 0$

(b) Увеличим поток f на найденом пути на x и изменим остаточную сеть соответствующим образом.

$|f|$ увеличивается на целое положительное число и ограничен сверху, значит алгоритм закончится. Корректен по теореме Форда-Фалкерсона. Асимптотика — $O(\text{ans} \cdot |E|)$.

Вопрос 8

Алгоритм Эдмонса-Карпа поиска максимального потока.

Единственное отличие от Форда-Фалкерсона в том, что берём не произвольный путь, а кратчайший. То есть используем bfs , а не dfs .

Асимптотика.

В таком случае итераций не больше чем $O(n \cdot m)$, тогда итоговая асимптотика $O(n \cdot m^2)$.

Доказательство корректности

Корректен ввиду корректности алгоритма Форда-Фалкерсона.

Вопрос 9

Блокирующий поток f — поток в сети G такой, что в G (но не обязательно в G_f) нет пути из s в t , вдоль которого можно протолкнуть поток.

Расстояние $dist(u, v)$ — минимальное расстояние из u в v по количеству ребёр.

Слоистая сеть — сеть, построенная из сети G следующим образом: разобьём V на

множества $V_i = \{v \in V \mid dist(s, v) = i\}$ и оставим ребра только между V_i и V_{i+1} (для всех i).

Процедура поиска произвольного блокирующего потока.

Идея проста, будем пускать поток пока получается, а для оптимизации учтем тот факт, что если мы уже однажды пытались пойти вдоль ребра, и узнали что после прохода по нему дойти до t не получится, то нам не стоит идти по этому ребру снова. Это аналог метки *used* в DFS. Асимптотика

$O(n \cdot m)$.

Алгоритм Диница поиска максимального потока.

Пока не найден *maxflow*:

1. Построить слоистую сеть по остаточной сети.
2. Пустить в ней произвольный блокирующий поток.

Корректность.

Теорема Форда-Фалкерсона учит нас что поток максимален \iff в остаточной сети нет увеличивающего пути. Если наш алгоритм завершился, то есть не сумел пустить блокирующий поток в слоистой сети, значит в слоистой сети нет пути из s в t . Но поскольку она содержит в себе все кратчайшие пути из истока в остаточной сети, это в свою очередь означает, что в остаточной сети нет пути из истока в сток.

Асимптотика.

$O(n^2 \cdot m)$.

Вопрос 10

Теоремы Карзанова. Обозначения

$$\bullet c_{out}(v) = \sum_{u \in V} c(v, u)$$

$$\bullet c_{in}(v) = \sum_{u \in V} c(u, v)$$

Общий потенциал сети $P = \sum_{v \in V} \min(c_{out}(v), c_{in}(v))$.

Теорема 1.

Количество итераций в алгоритме Диница $O(\sqrt{P})$

Теорема 2.

Если C — ограничение сверху на все *cap* ($\forall u, v \in V \ cap(u, v) \leq C$), то количество итераций в алгоритме Диница есть $O(C^{\frac{1}{3}} \cdot |V|^{\frac{2}{3}})$.

Единичная сеть — сеть, в которой все *cap* равны 1.

Асимптотика алгоритма Диница на единичных сетях.

Так как C из второй теоремы Карзанова равно 1, а потенциал сверху оценивается $|E|$, количество итераций есть $O(\min(\sqrt{|E|}, |V|^{\frac{2}{3}}))$.

Алгоритм поиска блокирующего потока будет работать за $O(|E|)$, т.к. каждое ребро не будет просмотрено более одного раза, т.к. каждое ребро уже при первом посещении будет либо проигнорировано, либо насытиться.

А значит асимптотика алгоритма — $O(\min(|E| \cdot \sqrt{|E|}, |E| \cdot |V|^{\frac{2}{3}}))$.

Алгоритм Хопкрофта-Карпа. (максимальное паросочетание в двудольном графе через максимальный поток)

Построим следующую сеть: Возьмём фиктивные s и t . Обозначим α и β — доли графа. Из s проведём рёбра *cap* = 1 к каждой вершине α . Затем из

каждой вершины β проведём ребро $cap = 1$ в t . Исходные ребра оставим на месте, направив их из α в β , их $cap = 1$. На этой сети найдем максимальный поток Диницем. Вершины, по которым прошел поток, будут лежать в паросочетании. Асимптотика: $O(E \cdot \sqrt{V})$ по первой теореме Карзанова ($\forall u \in \alpha c_{in}(u) = 1, \forall v \in \beta c_{out}(v) = 1$).

Вопрос 11

Задача поиска k — потока минимальной стоимости (постановка). Постановка похожа на постановку задачи о потоках. Только теперь у каждого ребра появляется цена. И мы платим штраф за поток f через ребро с ценой p равную $|f| \cdot p$. Требуется найти поток величины k минимальной стоимости.

Обратная сеть.

Считаем, что в нашем графе нет обратных рёбер, тогда проведём обратные рёбра к рёбрам с ценой p с $cap = 0, price = -p$. Такая сеть называется обратной.

Алгоритм поиска k -потока минимальной стоимости

Пока $(k - -) \neq 0$:

1. В остаточной сети найти кратчайший по $price$ путь из s в t .
2. Пустить единицу потока вдоль этого пути.
3. Перестроить остаточную сеть.

Условие применимости:

В изначальном цикле нет циклов отрицательного веса.

При таком условии можем искать кратчайший по $price$ путь с помощью алгоритма Форда-Беллмана:

$d[k][u]$ — количество путей длины k рёбер, заканчивающихся в вершине u . Опустим в этом массиве длину, т.е. сделаем его одномерным.

```
bool fordBellman(s):
    for v ∈ V
        d[v] = 1
    d[s] = 0
    for i = 0 to |V| - 1
        for (u, v) ∈ E
            if d[v] > d[u] + ω(u, v) // ω(u, v) - вес
                d[v] = d[u] + ω(u, v)
    for (u, v) ∈ E
        if d[v] > d[u] + ω(u, v)
            return false
    return true
```

Лемма.

Если f и g — потоки, то $\exists h$ — поток такой, что $f = g + h$ (порёберно).

Доказательство

Нетрудно проверяется, что $h = f - g$ (порёберно) является потоком.

Утверждение.

Если f — $\text{mincost } k\text{-flow}$, а p — кратчайший по price путь из s в t в G_f , то $g = f + p$ — $\text{mincost } (k+1)\text{-flow}$.

Доказательство

Пусть m — произвольный $(k+1)$ -flow. Тогда h — поток такой, что $m = f + h$. Декомпозирируем (представим в виде совокупности путей из s в t и циклов, все имеющие положительный поток) h в пути p_i и циклы c_j . Тогда $\forall j \text{ price}(c_j) \geq 0$, иначе $f + c_j$ — k -flow меньшей стоимости. Также $\forall i \text{ price}(p_i) \geq \text{price}(p)$, так как p — кратчайший по price путь. Тогда $\text{price}(m) = \text{price}(f) + \text{price}(h) = \text{price}(f) + \sum \text{price}(p_i) + \sum \text{price}(c_j) \geq \text{price}(f) + \text{price}(p) = \text{price}(g)$.

Теорема (Следствие).

После i -ой итерации алгоритм выше находит $\text{mincost } i\text{-flow}$.

Лемма об отсутствии отрицательных циклов

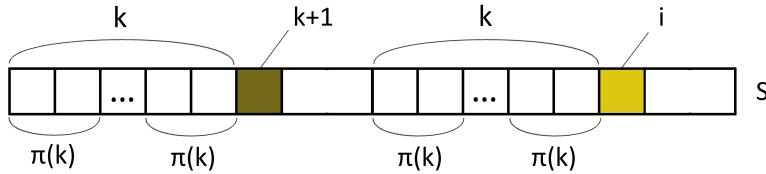
Если в G нет циклов отрицательного веса, а G_f — остаточная сеть после пропуска единицы вдоль кратчайшего пути из s в t пути в G , то в G_f нет циклов отрицательного веса.

Вопрос 12

Префикс-функция от строки s — массив π , в котором $\pi[i]$ — длина наибольшего собственного (не совпадающего со всей строкой) суффикса $s[:i]$, который совпадает с префиксом $s[:i]$.

Построение.

- Заметим, что $\pi[i+1] \leq \pi[i] + 1$ (удалим из суффикса, оканчивающегося на позиции $i+1$ и имеющего длину $\pi[i+1]$ последний символ — получим суффикс, оканчивающийся на позиции i и имеющий длину $\pi[i+1] - 1$, следовательно неравенство $\pi[i+1] - 1 > \pi[i]$ неверно).
- Если $s[i] == s[\pi[i-1]]$, то $\pi[i] = \pi[i-1] + 1$.
- Если $s[i] \neq s[\pi[i-1]]$, то нужно попытаться попробовать суффикс меньшей длины. Хотелось бы сразу перейти к следующему по величине несобственному суффиксу $s[:i]$, который совпадает с префиксом $s[:i]$. Найдём его длину k следующим образом: положим $k = \pi[i-1]$. Пока $k > 0$: в случае, когда символы $s[k+1]$ и $s[i]$ не совпадают, то $k := \pi[k]$, иначе $\pi[i] := k + 1$. Если $k = 0$, то $\pi[i] = 1$ при $s[i] = s[0]$ и $\pi[i] = 0$ иначе .



Асимптотика.

Заметим, что каждая итерация перебора k означает уменьшение значения π . Однако за итерацию всего алгоритма (подсчёт следующего значения) π не может увеличиться более чем на единицу. Значит в целом алгоритм имеет сложность $O(n)$.

Вопрос 13

Z-функция от строки S и позиции x — это длина максимального префикса подстроки, начинающейся с позиции x в строке S , который одновременно является и префиксом всей строки S (назовём это **Z-блоком**). Более формально, $Z[i](s) = \max_{s[i \dots i+k] = s[0 \dots k]} k$. Значение Z — функции от первой позиции не определено, поэтому его обычно приравнивают к нулю или к длине строки.

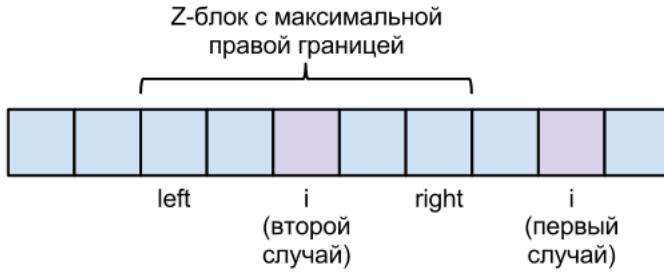
Самый правый Z-блок [L; R] — среди найденных Z — блоков тот, правая граница которого максимальна.

Построение.

Пусть нам известны значения Z — функции от 0 до $i - 1$. Найдём $Z[i]$. Рассмотрим два случая.

1. $i > R$: Находим j — первую позицию в строке S для которой $S[i+j] \neq S[j]$, тогда j это значение Z — функции для позиции i . Тогда необходимо обновить $L = i$ и $R = i + j - 1$. В данном случае будет определено корректное значение $Z[i]$ в силу того, что оно определяется наивно.
2. $i \leq R$:

- Если $Z[i - L] + i \geq R$, то есть Z -блок, начинающийся в $i - L$ выходит за пределы префикса, соответствующего текущему самому правому Z -блоку, то $Z[i] \geq R - i$ и тогда нужно наивно досчитать значение $Z[i]$, обновив соответственно L и R .
- Если $Z[i - L] + i < R$, то есть Z -блок, начинающийся в $i - L$ полностью содержится в префиксе, соответствующем текущему самому правому Z -блоку, не захватывая его последний символ, то $Z[i] = Z[i - L]$, иначе мы могли бы увеличить значение $Z[i - L]$.



Асимптотика

Этот алгоритм работает за $O(|S|)$, так как каждое наивное действие сдвигает самый правый Z -блок, а это нельзя сделать более, чем $|S|$ раз.

Вопрос 14

Полиномиальный хэш h — число $h = \text{hash}(s[0..n - 1]) = (p^{n-1} \cdot s[0] + \dots + p^0 \cdot s[n - 1]) \% M$, где p, M — некоторые простые числа, а $s[i]$ — код i -ого символа строки s .

Проверка подстрок данной строки s на равенство с помощью хешей за $O(1)$ на запрос и $O(|s|)$ предподсчёта.

Сделаем предподсчёт степеней p и $h(s[0 : i])$. Предподсчёт можно делать по схеме Горнера $h(s[0 : n]) = (((((a_0 \cdot p + a_1) \cdot p + a_2) \cdot p + \dots) \cdot p + a_{n-1})$. Тогда $h(s[l : r]) = h(s[0 : r]) - h(s[0 : l]) \cdot p^{r-l+1}$

Тогда при запросе на сравнение подстрок можно сравнивать не подстроки, а их хэши.

Предпосчёт занял $O(|s|)$, ответ на запрос будет $O(1)$.

Вопрос 15

Поиск наибольшего общего префикса двух подстрок данной строки

с помощью хешей за $O(\ln(\max(|s_1|, |s_2|)))$.

Бинарный поиск по длине префикса i . Считаем $h(s1[0 : i])$ и $h(s2[0 : i])$.

Проверяем их на равенство и бинпоиском находим ответ. Хеш считается за $O(\max(|s1|, |s2|))$, базовые операции также работают за $O(1)$. Получаем асимптотику $O(\ln \max(|s1|, |s2|))$.

Вопрос 16

Алгоритм Рабина-Карпа поиска подстроки в строке.

s, n — исходная строка и ее длина, w, m — искомая подстрока и ее длина.

- Подсчитать $h(s[0 \dots m - 1])$ и $h(w[0 \dots m - 1])$, а также p^m , для ускорения ответов на запрос.
- Для $i \in [0 \dots n - m]$ вычислить $h(s[i \dots i + m - 1])$ и сравнить с $h(w[0 \dots m - 1])$. Если используется полиномиальный хэш, то $h(s[i \dots i + m - 1]) = (p \cdot h(s[i - 1 \dots i + m - 2]) - p^m \cdot s[i - 1] + s[i + m - 1]) \% M$.

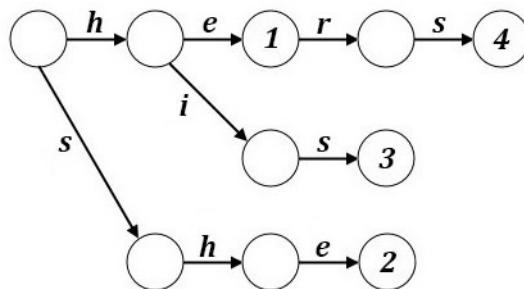
Если хэши равны, то w скорее всего содержится в строке s начиная с позиции i , хотя возможны и ложные срабатывания алгоритма.

Асимптотика.

Изначальный подсчёт хешей выполняется за $O(m)$. Каждая итерация выполняется за $O(1)$, в цикле всего $n - m + 1$ итераций. Итоговое время работы алгоритма $O(n + m)$. Однако, если требуется исключить ложные срабатывания алгоритма полностью, то придется проверить все полученные позиции вхождения на истинность и тогда в худшем случае итоговое время работы алгоритма будет $O(n \cdot m)$.

Вопрос 17

Бор — структура данных для хранения набора строк, представляющая из себя взвешенное дерево с символами на рёбрах. Строки получаются последовательной записью всех символов, хранящихся на рёбрах между корнем бора и терминальной вершиной. Бор для набора образцов $\{he, she, his, hers\}$:



Способы хранения переходов из вершины.

1. Массив $Node^*$ размера $|\Sigma|$ в вершине, что оптимально по времени, но не оптимально по памяти.
2. Хэш-таблица $Node^*$, что довольно оптимально по времени (амортизированная константа), много более оптимально по памяти.

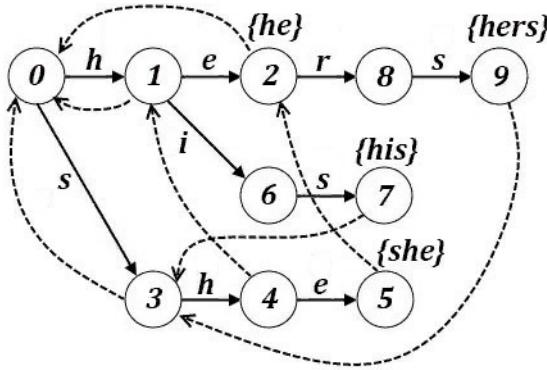
Вопрос 18

Сортировка строк с помощью бора.

1. Сложить все строки в бор.
2. Запустить в боре dfs , приоритезирующий переходы по символам с меньшим численным значением.

Вопрос 19

Суффиксная ссылка $\pi(v)$ для вершины v — это вершина, в которой оканчивается длиннейший собственный суффикс строки, соответствующей вершине v . Для корня суффиксная ссылка ведёт в себя.



Узлы бора можно понимать как состояния автомата, а корень как начальное состояние. Узлы бора, в которых заканчиваются строки, становятся терминальными состояниями.

Тогда рёбра бора — переходы в автомате по соответствующей букве. Однако одними только рёбрами бора нельзя ограничиваться. Если пытаться выполнить переход по какой — либо букве, а соответствующего ребра в боре нет, то тем не менее нужно перейти в какое-то состояние. Для этого и нужны суффиксные ссылки.

Для переходов по автоматау заведём в узлах функции:

- $\text{parent}(u)$ — возвращает родителя вершины u .
- Функция перехода

$$\text{jump}(u, c) = \begin{cases} v, & \text{если } v \text{ — ребёнок } u \text{ по символу } c, \\ \text{root}, & \text{если } u \text{ — корень и у } u \\ & \text{нет ребёнка по символу } c, \\ \text{jump}(\pi(u), c), & \text{иначе.} \end{cases}$$

Обозначим за $[v]$ слово, приводящее в вершину v в боре. Тогда заметим, что $\pi(v) = \text{jump}(\pi(\text{parent}(v)), c)$, где c — последний символ $[v]$. Из этого следует, что функции перехода и суффиксные ссылки можно найти либо алгоритмом обхода в глубину с ленивыми вычислениями, либо с помощью алгоритма обхода в ширину.

Сжатая терминальная ссылка $\text{term}(v)$ вершины v — первая терминальная вершина при переходе из v по суффиксным ссылкам:

$$\text{term}(u) = \begin{cases} \pi(u), & \text{если } \pi(u) \text{ — терминальная.} \\ \emptyset, & \text{если } \pi(u) \text{ — корень.} \\ \text{term}(\pi(u)), & \text{иначе.} \end{cases}$$

Аналогично обычным суффиксным ссылкам сжатые суффиксные ссылки могут быть найдены при помощи ленивой рекурсии.

Вопрос 20

Алгоритм Ахо-Корасик.

Алгоритм поиска подстрок из словаря в тексте:

- Построить бор с суффиксными и терминальными ссылками на заданном словаре.
- Встать в корень бора $cur = root$. Для каждого символа c текста:
 1. $cur := \delta(cur, c)$.
 2. Засвидетельствовать вхождение строк, соответствующих вершинам $cur, term(cur), term(term(cur)) \dots root$, если эти вершины терминальны.

Сложность алгоритма. Пусть Sum — суммарная длина подстрок в наборе, N — длина текста, по которому осуществляется поиск, Ans — число найденных вхождений.

1. Построение бора требует $O(Sum)$
2. Построение суффиксных ссылок, по сути, BFS по дереву, требует $O(Sum)$
3. Построение сжатых терминальных ссылок, также BFS по дереву, требует $O(Sum)$
4. Цикл по каждому из символов — $O(N)$
5. Проходы по сжатым терминальным ссылкам суммарно займут $O(Ans)$

Итого, имеем $O(Sum + N + Ans)$

Вопрос 21

Поиск всех вхождений слов из словаря V в текст T .

- Алгоритм Ахо-Карасик с асимптотикой $O(|V| + |T| + |ans|)$.
- Алгоритм Рабина — Карпа с асимптотикой $O(|V| \cdot (|T| + \max_{S \in V} |S|))$.
- Алгоритм Кнута — Морриса — Пратта(префикс — функция) с асимптотикой $O(|V| \cdot (|T| + \max_{S \in V} |S|))$. Пусть t — текст, s — искомая подстрока. Образуем строку $s + \# + t$, где символ $\#$ — это разделитель, который не должен нигде более встречаться. Посчитаем для этой строки префикс-функцию. Теперь рассмотрим её значения, кроме первых $n + 1$ (которые, как видно, относятся к строке s и разделителю). По определению, значение $\pi[i]$ показывает найденнейшую длину подстроки, оканчивающейся в позиции i и совпадающей с префиксом. Но в нашем случае это $\pi[i]$ — фактически длина наибольшего блока совпадения со строкой s и оканчивающегося в позиции i . Больше, чем n , эта длина быть не может — за счёт разделителя. А вот равенство

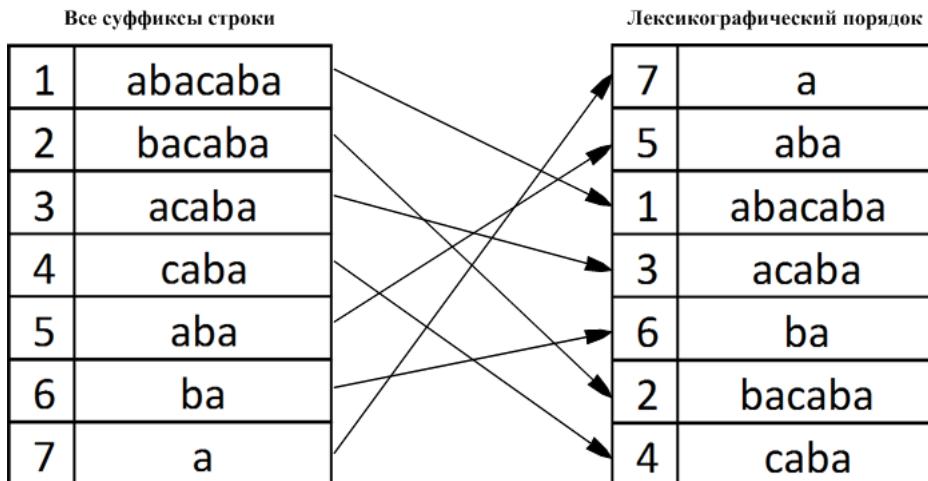
$\pi[i] = n$ (там, где оно достигается), означает, что в позиции i оканчивается искомое вхождение строки s (только не надо забывать, что все позиции отсчитываются в склеенной строке $s + \# + t$). Таким образом, если в какой-то позиции i оказалось $\pi[i] = n$, то в позиции $i - (n+1) - n + 1 = i - 2 \cdot n$ строки t начинается очередное вхождение строки s в строку t .

- Аналогично предыдущему пункту работает алгоритм с Z — функцией.

Подсчёт количества вхождений слов из словаря. Аналогично

Вопрос 22

Суффиксный массив строки $s[0 \dots n]$ — массив suf целых чисел от 1 до n , такой, что суффикс $s[suf[i] \dots n]$ — i -й в лексикографическом порядке среди всех непустых суффиксов строки s . Пример для строки $s = abacaba$:



Построение с помощью хешей.

Пусть нам необходимо сравнить два суффикса s . Найдем сначала их наибольший общий префикс двоичным поиском по его длине с проверкой на равенство хэшами.

Дальнейшее сравнение суффиксов можно произвести по первому символу после общего префикса (или понять, что более короткий суффикс меньше, если он оказался общим префиксом).

Тогда сравнение двух суффиксов работает за $O(\log n)$, а значит сортировка может быть произведена за $O(n \log^2 n)$.

Вопрос 23.

Задача LCP (постановка).

Найти lcp — длину наибольшего общего префикса для двух суффиксов строки.

Во время построения суффиксного массива сохраним классы эквивалентности, тогда проверить две подстроки на равенство можно за $O(\log n)$: если

их классы эквивалентности на k -ом шаге равны (т.е. первые 2^k символов равны), то надо откусить 2^k первых символов этих префиксов и продолжить искать наибольшие общие префиксы у оставшейся части, иначе k разделить на 2 и начать сначала.

Вопрос 24

Сведение задачи lcp произвольных суффиксов к lcp в суффиксном массиве.

Очевидно, что

$$\begin{aligned} lcp(i, j) &= \min(lcp(where[i], where[i] + 1), \\ &lcp(where[i] + 1, where[i] + 2), \dots, \\ &lcp(where[j] - 1, where[j])) \end{aligned}$$

Где $where$ — обратная к suf перестановка (б.о.о. предполагается, что $where[i] < where[j]$). Ответ на запрос этой величины можно возвращать за $O(1)$ и $O(n \cdot \log n)$ предподсчёта (*sparse table*).

Доказательство.

Суффиксы в суффиксном массиве отсортированы по возрастанию, а значит неравенство выполняется и между их префиксами. Из этого понятно, что $lcp(i, j)$ не может быть больше, чем указанная величина.

С другой стороны очевидно, что указанная величина задаёт длину общего префикса i и j , то есть соблюдено неравенство и в другую сторону.

Если непонятно, рассмотреть суффиксный массив явно.

Вопрос 25

Суффиксное дерево T для строки s — дерево с $n = |s|$ листьями, обладающее следующими свойствами:

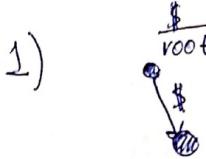
- Каждая внутренняя вершина дерева имеет не меньше двух детей.
- Каждое ребро помечено непустой подстрокой строки s .
- Никакие два ребра, выходящие из одной вершины, не могут иметь пометок, начинающихся с одного и того же символа.
- Дерево должно содержать все суффиксы строки s , причем каждый суффикс заканчивается точно в листе и нигде кроме него.

Данное определение порождает следующую проблему: рассмотрим дерево для строки $habxa$: суффикс ha является префиксом суффикса $habxa$, а, значит, этот суффикс не заканчивается в листе. Для решения проблемы в конце строки s добавляют символ, не входящий в исходный алфавит: защитный символ. Обозначим его как $\$$. Любой суффикс строки с защитным символом действительно заканчивается в листе и только в листе, т. к. в такой строке не существует двух различных подстрок одинаковой длины, заканчивающихся на $\$$.

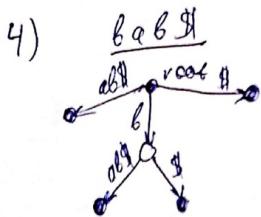
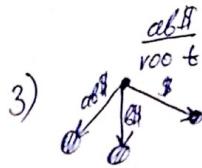
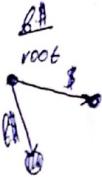
Хранить подстроки выгоднее индексами, получаем память $O(n)$.

Пример построения:

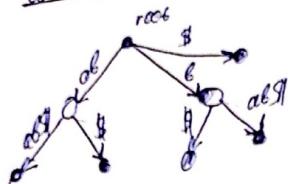
$S = abaaabab\$$



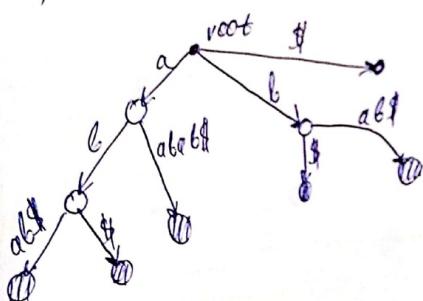
2)



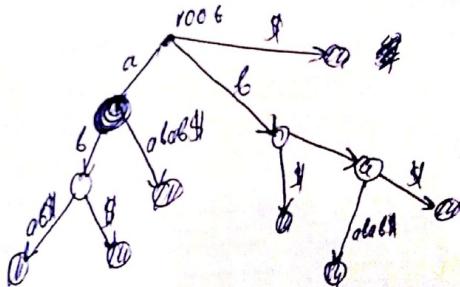
5) abab\\$



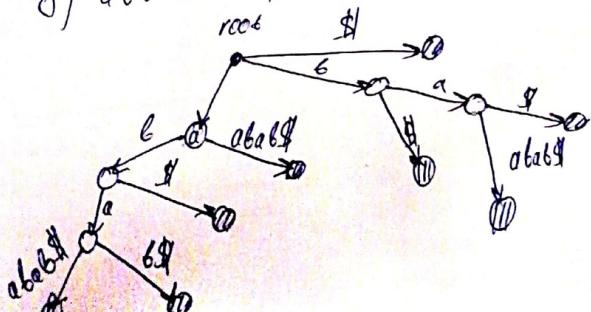
6) aa bab\\$



7) baa bab\\$



8) abaaabab\\$



$\underline{\mathcal{O}(n^2)}$

Вопрос 26

Поиск вхождения s в t по суффиксному дереву строки t .
Алгоритм.

- Построить суффиксное дерево для t .
- Встать в корень $cur = root$. Для каждого символа c строки s :
 1. Спуститься из cur по c и обновить cur , если это возможно.
 2. Иначе завершить алгоритм: s не входит в t как подстрока.
- Завершить алгоритм: s входит как подстрока в t .

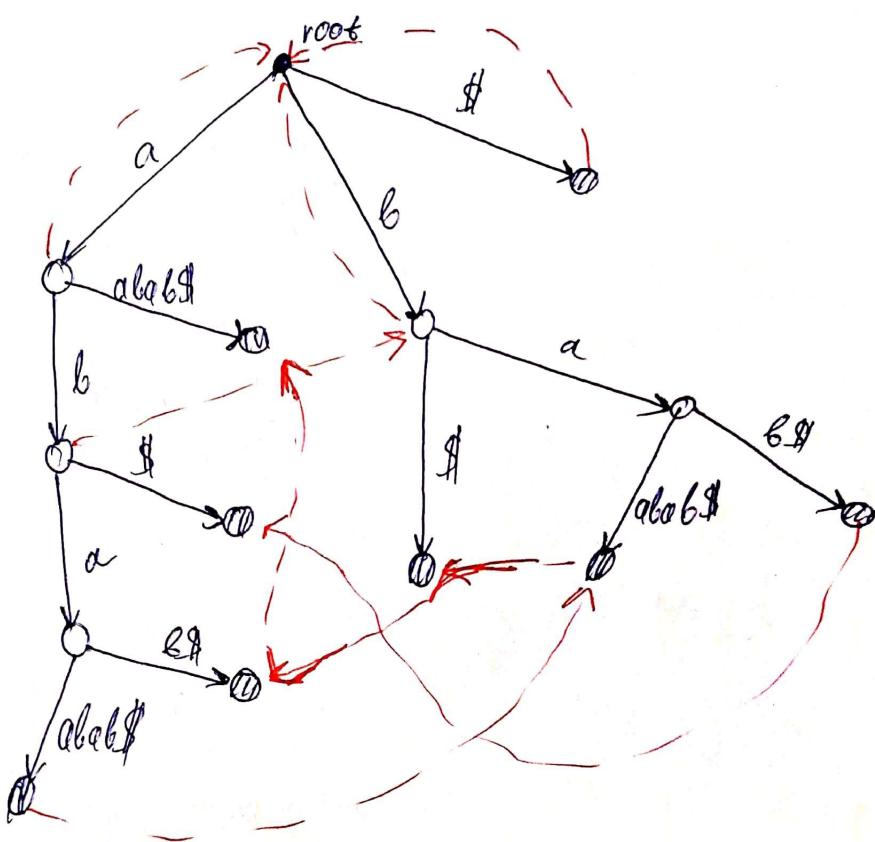
Асимптотика. Предподсчёт за $O(|t|)$ и $O(|s|)$ на один запрос.

Вопрос 27

Суффиксные ссылки.

В суффиксном дереве суффиксная ссылка строки $s[i : j]$ ведёт в строку $s[i + 1 : j]$. Пример:

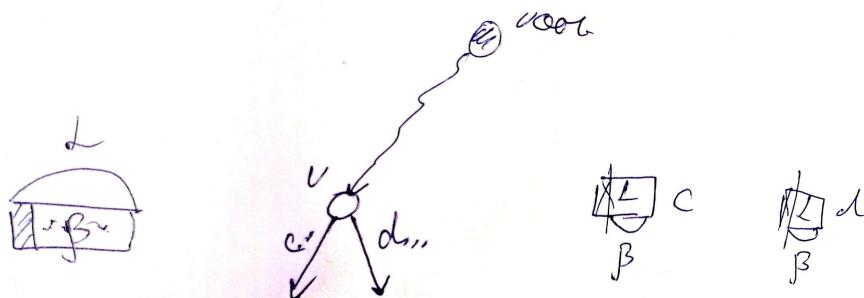
$$S = abaabab \$$$



Утверждение. Суффиксная ссылка не может вести в “середину” ребра.
Доказательство.

1. Терминальная вершина \Rightarrow суффиксная ссылка ведёт в терминалную, т.к. суффикс суффикса — суффикс.
2. Не терминальная вершина \Rightarrow суффиксная ссылка ведёт в не терминалную или корень. Докажем это.

Пусть есть не терминальная вершина v , ей соответствует какая-то строка α . Так как v не терминальная, то из неё идут как минимум два ребра со строками $c\dots$ и $d\dots$. Тогда в исходном тексте можно продлить строку α буквами c и d . Отбросив первую букву строки α , получим строку β . Но тогда и строку β можно продлить буквами c и d . Значит, мощность множества детей суффиксной ссылки v больше равна двум. Тогда суффиксная ссылка не ведёт в середину ребра.

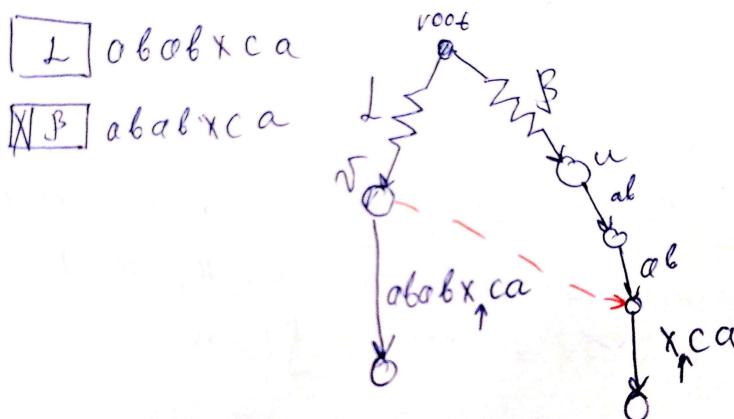


Вопрос 28

Процедура поиска суффиксной ссылки для подстроки, заданной вершиной или положением на ребре.

Чтобы найти суффиксную ссылку, достаточно перейти в суффиксную ссылку родителя и перейти из неё по тем символам, которые написаны на ребре, ведущем в данную вершину (или их части, если нужно найти суффиксную ссылку для середине ребра).

Если же родитель является корнем, то нужно из него перейти по символам, написанным на ребре (или их части), предварительно отбросив первый.



Вопрос 29

Алгоритм Укконена.

Построение суффиксного дерева для строки s путём последовательного построения суффиксных деревьев для её префиксов. Заметим, что если имеется суффиксное дерево для строки α , то суффиксное дерево для строки αc можно получить, добавив ко всем суффиксам символ c . Суффиксы рассматриваемого на текущей итерации префикса можно разбить на три группы:

1. Листья.
2. Не листья, из которых нет перехода по c .
3. Не листья, из которых есть переход по c .

Они разбиваются на три непрерывные группы. Действительно, пусть у нас есть самый короткий лист. Тогда все большие терминальные вершины так же листья, поскольку если это не так, то по ним есть несколько переходов, а из них суффиксная ссылка ведет в наш лист. Тогда и у нашего листа есть несколько переходов. Противоречие. Аналогично можно рассмотреть самый длинный нелист, у которого есть переход по c . Заметим, что

1. При добавлении c к листу он останется листом. Значит вместо того, чтобы добавлять по одному символу к листу, можно сразу завершить его всей оставшейся строкой и более не рассматривать.
2. Если из не листа нет перехода по c , то придётся создать ветвление и тогда снова получится лист, а значит его тоже можно сразу завершить.
3. Если из не листа есть переход по c , то по нему можно просто перейти и сделать следующую вершину терминальной. В добавок, по свойству суффиксных ссылок, для всех более коротких суффиксов переход по c тоже существует.

Алгоритм.

Храним ссылку на самый длинный суффикс, не являющийся листом. Изначально $cur = root$. Для каждого символа c строки s :

1. Пока нет перехода из cur по c , производим действия, описанные в первых двух пунктах и обновляем cur , переходя по суффиксной ссылке.
2. Обновляем cur , спускаясь по c .

У всех создаваемых вершин, кроме терминальных, насчитываем суффиксную ссылку с помощью процедуры сразу при создании.

Ассимптотика.

Алгоритм линеен — $O(|s|)$.

Вопрос 30

Детерминированный конечный автомат A — орграф, на рёбрах которого написаны буквы. Какая-то вершина отмечена стартовой. А любая вершина может быть обозначена терминалной. Детерминированность гарантирует, что все символы написанные на рёбрах из одной вершины различны.

Автомат принимает слово s , если встав в стартовую вершину и читая буквы из s можно достичь терминалной вершину.

Минимальный автомат, содержащий все суффиксы s и только их — детерминированный автомат с минимальным количеством вершин, принимающий суффиксы s и только их.

Языком автомата A называется $L(A) = \{s \mid s \text{ принимается } A\}$.

Правый контекст слова u в языке L — $R_L(u) = \{v \mid uv \in L\}$.

Отношение эквивалентности относительно языка L : $u \sim_L v \iff R_L(u) = R_L(v)$.

Пусть дана строка s и язык $L = \{s^1, s^2, \dots, s^n, \varepsilon\}$ (все суффиксы s). Обозначим $R_s(x) = \{z \mid xz \in L \text{ т.е. } xz \text{ — суффикс } s\}$

Вопрос 31

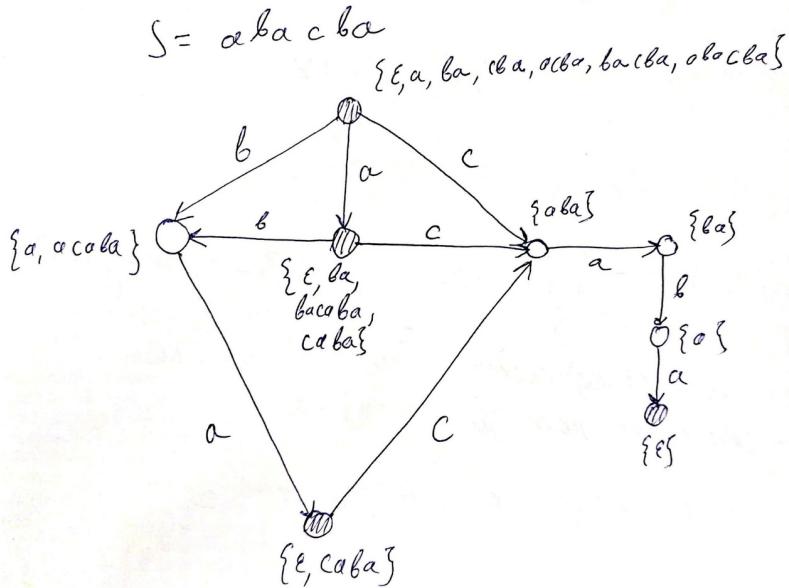
Теорема Майхилла-Нероуда.

Для языка L множество всевозможных правых контекстов $\{R_L(u) \mid u \in \Sigma^*\}$ конечно \iff язык L — автоматный и его минимальный автомат и имеет $|\{R_L(u) \mid u \in \Sigma^*\}|$ состояний.

Вопрос 32

Суффиксный автомат — минимальный детерминированный конечный автомат, который принимает все суффиксы строки и только их.

Утв. Вершины в автомате соответствуют классам эквивалентности по введённому отношению.



Сканировано с CamScanner

/*

Утв. Количество вершин в суффиксном автомате строки s длины n — не больше $2n - 1$.

Утв. Количество рёбер в суффиксном автомате строки s длины n — не больше $3n - 4$.

Приечания:

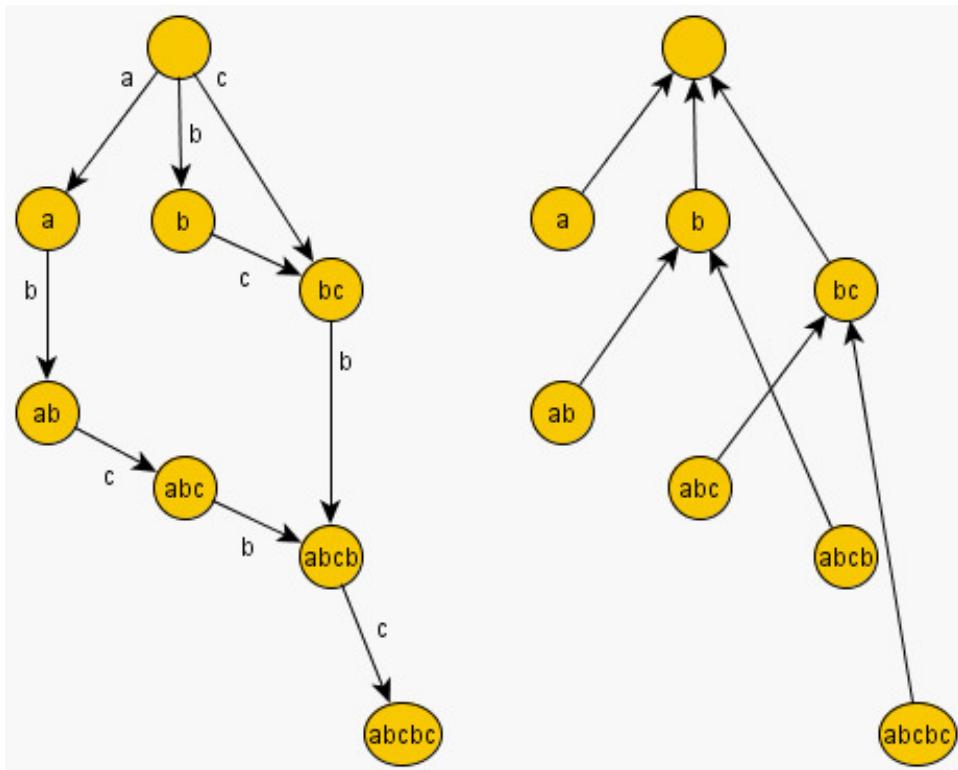
1. Суффиксный автомат ациклический граф.
2. Количество вхождений t в s — $|R_s(t)|$;
3. Первое и последнее вхождение t в s — самый длинный и самый короткий пути до терминального состояния соответственно.
4. Найти наибольшую общую подстроку двух строк s и t : построить суффиксный автомат по строке $s \# t$. И если из какой-то вершины можно добраться до терминала прочитав решётку и не прочитав решётку, это и означает, что эта вершина соответствует общей подстроке.

*/

Вопрос 33

- $\text{longest}([u])$ — самая длинная строка в классе эквивалентности $[u]$.
- $\text{shortest}([u])$ — самая короткая строка в классе эквивалентности $[u]$.

Суффиксная ссылка состояния, соответствующего классу эквивалентности слова α $\text{link}([\alpha])$ ведет в состояние, соответствующее самому длинному суффиксу $\text{longest}([\alpha])$. Ниже пример суффиксных ссылок в автоматае, построенном для строки “abcabc”.



Вопрос 34

В суффиксном автомате *longest* — самый длинный представитель своего класса.

Критерий того, что строка является *longest* в своём классе.

Строка $\alpha = \text{longest}([\alpha]) \iff \begin{cases} \alpha \text{ — префикс } s \\ \exists a \neq b \text{ — символы : } a\alpha, b\alpha \text{ — подстроки } s \end{cases}$

Доказательство — 0.2.13

Вопрос 35

Построение суффиксного автомата.

Будем обозначать длину $\text{longest}(q)$, как $\text{len}(q)$. Будем добавлять символы строки s по одному, перестраивая при этом автомат. Изначально автомат состоит из одного состояния root , для которого $\text{len}(\text{root}) = 0$, а $\text{link}(\text{root}) = \text{null}$.

Обозначим состояние last , соответствующее текущей строке до добавления нового символа c (изначально $\text{last} = \text{root}$). Создадим новое состояние cur , $\text{len}(\text{cur}) = \text{len}(\text{last}) + 1$. Рассмотрим переход из last по текущему символу c . Если перехода нет, то добавляем переход в cur , переходим по суффиксной ссылке и повторяем процедуру снова и снова. Если переход существует, то останавливаемся и обозначим текущее состояние за has__c . Если перехода не нашлось и по суффиксным ссылкам мы дошли до null , то $\text{link}(\text{cur}) = 0$.

Допустим, что мы остановились в состоянии has__c , из которого существует переход с символом c . Обозначим состояние, куда ведёт переход,

через $has_c_to_c$. Рассмотрим два случая:

1. Если $\text{len}(has_c) + 1 = \text{len}(has_c_to_c)$, то $\text{link}(cur) = has_c_to_c$.
2. В противном случае, создадим новое состояние new , скопируем в него $has_c_to_c$ вместе с суффиксными ссылками и переходами. $\text{len}(new)$ присвоим значение $\text{len}(has_c) + 1$. Перенаправим суффиксную ссылку из $len_c_to_c$ в new и добавим ссылку из cur в new . Пройдём по всем суффиксным ссылкам из состояния has_c и все переходы в состояние $has_c_to_c$ по символу c перенаправим в new .

Асимптотика: $O(|S|)$.

Вопрос 36

Постановка задачи про хеш-таблицы:

Есть некоторое множество, хотим делать три вещи:

1. **Add(k, v)** (ключ, значение);
2. **Remove(k, v)**;
3. **Find(k, v) $\rightarrow v$.**

где $k \in \mathbb{U}$. Все операции хотим за константу.

Direct address.

Простой массив, индексы — ключи: Предполагаем, что все элементы можно пронумеровать от 0 до $|\mathbb{U}| - 1$. Тогда можем создать массив размера $|\mathbb{U}|$, заполненный 0 для ключей. При добавлении ставим 1 в соответствующую ячейку и элемент v , при удалении — 0 для ключа. При **Find** смотрим в ячейку, соответствующую ключу k возвращаем то, что лежит в ячейке для значения.

Хеш-функция — функция $h : \mathbb{U} \longrightarrow \mathbb{Z}_M$.

Работаем также как и раньше, только массив размера не $|U|$, а M .

Коллизия — $x \neq y : h(x) == h(y)$.

Разрешение коллизий с помощью цепочек.

Каждая ячейка i массива содержит указатель на начало списка всех элементов, хеш-код которых равен i , либо указывает на их отсутствие. Коллизии приводят к тому, что появляются списки размером больше одного элемента.

В зависимости от того нужна ли нам уникальность значений операции вставки у нас будет работать за разное время. Если не важна, то мы используем список, время вставки в который будет в худшем случае равна $O(1)$. Иначе мы проверяем есть ли в списке данный элемент, а потом в случае его отсутствия мы его добавляем. В таком случае вставка элемента в худшем случае будет выполнена за $O(n)$.

Вопрос 37

Хэш функция — функция $h : \mathbb{U} \rightarrow \mathbb{Z}_m$.

Коллизия — ситуация $x \neq y$, но $h(x) = h(y)$.

Пространство хэш функций — $\mathbb{H} \subset \mathbb{Z}_M^{\mathbb{U}}$.

Simple Uniform Hashing (SUH) гипотеза.

Хэш функция выбирается из \mathbb{H} случайно равномерно, то есть её значение на каждом ключе выбирается равномерно и независимо.

Теорема

В предположении SUH , при разрешении коллизий методом цепочек, математическое ожидание длины цепочки — $O\left(\frac{N}{M}\right)$, где N — число, добавленных ключей, а $\frac{N}{M} = \alpha$ — *load factor*.

Доказательство

Длина цепочки $j = |c_j|$.

$$\begin{aligned}|c_j| &= \sum_{i=1}^N \mathbb{I}(h(k_i) == j) \quad (\mathbb{I} — индикатор, по сути if) \\ E(|c_j|) &= \sum_{i=1}^N P(h(k_i) == j) = \sum_{i=1}^N \frac{1}{M} = O\left(\frac{N}{M}\right)\end{aligned}$$

Вопрос 38

Universal Hashing Family (UHF).

Семейство хэш функций $\mathbf{H} \subseteq \mathbb{H}$ — универсально, если при равномерном случайному выборе $h \in \mathbf{H}$

$$\forall x \neq y \in \mathbb{U} : P_h(h(x) = h(y)) = O\left(\frac{1}{M}\right) \left[\leq \frac{1}{M} \right]$$

Где вероятность берётся по всем хэш функциям семейства.

Если $= O\left(\frac{1}{M}\right)$, то семейство называется слабым универсальным семейством.

Если $\leq \frac{1}{M}$, то семейство называется сильным универсальным семейством.

Мат. ожидание длины цепочки фиксированного ключа k , в предположении универсальности семейства.

Обозн. N — количество всех элементов, c^k — цепочка ключа k . В предположении UHF и того, что $P(h(k) = h(k)) = 1$, получаем:

$$E(|c^k|) = \sum_{i=1}^N P(h(k_i) = h(k)) \leq 1 + O\left(\frac{N}{M}\right)$$

Вопрос 39

Зафиксируем $\mathbb{U} : \{0, \dots, p - 1\}$, p — простое, $p > M$.

Пример универсального семейства.

$\mathbb{H} = \{h_{\alpha, \beta} : \mathbb{U} \longrightarrow \mathbb{Z}_M \mid \alpha \in \{1, \dots, p - 1\}, \beta \in \{0, \dots, p - 1\}\}$, где $h_{\alpha, \beta}(x) = ((\alpha \cdot x + \beta) \bmod p) \bmod M$.

Вопрос 40

Совершенное хеширование (задача Fixed Set).

Из запросов оставляем только **Find** (убираем **Add** и **Remove**). Хотим:

- Иметь N фиксированных ключей.
- Иметь $O(N)$ памяти.
- Иметь $O(N)$ матожидание времени построения.
- Иметь $O(1)$ в худшем случае на запрос.

Идея в том, чтобы заменить цепочки (списки) на хеш-таблицы размера n_i^2 , где n_i — количество элементов, попавших в данную ячейку, так, чтобы в этих хеш-таблицах не было коллизий.

Алгоритм FKS Hashing.

1. Подбираем из универсального семейства хеш функций хорошую внешнюю хеш функцию.
2. Подбираем из универсального семейства хеш функций хорошую локальную хеш функцию в каждой ячейке.

Оказывается, что матожидание количества необходимых итераций подбора для достижения требований на каждом шаге константа. Поэтому в целом матожидание построения есть $O(N)$.

Вопрос 41

Открытое хеширование.

Хотим хранить *set* (только ключи, без значений). Заведём M ячеек в массиве *arr*. Введем функцию $run(x, i) \rightarrow \mathbb{N}$, где x — ключ, а i — номер попытки. Далее под *идем* будем подразумевать увеличение i на 1 и переход в ячейку $run(x, i)$.

1. Линейное пробирывание: $run(x, i) = (h(x) + i) \bmod M$;
2. Квадратичное пробирывание: $run(x, i) = (h(x) + i^2) \bmod M$;
3. Двойное хеширование: $run(x, i) = (h_1(x) + h_2(x) \cdot i) \bmod M$ для фиксированных h_1, h_2 .

То есть например при линейном пробировании мы будем посещать ячейки $h(x), h(x) + 1, h(x) + 2, \dots$

Ячейки таблицы будут иметь состояния:

1. Дырка — пустая ячейка.
2. Элемент — ячейка с элементом.
3. *TombStone* — ячейка, элемент которой был удалён.

Также, если хеш – таблица заполнена, то мы увеличиваем ее в два раза, выбираем новую функцию и все пересчитываем. Теперь опишем операции:

- **Add**

Идём до первой дырки или *TombStone*. Если на этом пути встретили наш элемент x (т.е. $arr[run(x, i)] = x$), то ничего не делаем, иначе:

1. Если дошли до дырки, то просто записываем наш элемент x в эту дырку.
2. Если дошли до *TombStone*, то запоминаем это место, и идём до дырки или нашего элемента x . Если встретили наш элемент, то ничего не делаем. Если дошли до дырки, то записали наш элемент в *TombStone*, который запомнили.

- **Remove**

Идём до первой дырки, игнорируя *TombStone*, если встретили наш элемент x , то удаляем его.

- **Find**

Идём до первой дырки, игнорируя *TombStone*, если встретили элемент, то он присутствует в множестве.

Вопрос 42

Определение k -независимого семейства.

Семейство хэш функций k -независимое, если

$\forall x_1, x_2, \dots, x_k \ h(x_1), h(x_2), \dots, h(x_k)$ — независимые в совокупности случайные величины ($\forall \alpha_i P((h(x_1) \leq \alpha_1) \cap (h(x_2) \leq \alpha_2) \cap \dots) = P(h(x_1) \leq \alpha_1) \cdot P(h(x_2) \leq \alpha_2) \cdot \dots$).

Пример k -независимого семейства.

$h(x) = (\alpha_1 \cdot x^{k-1} + \alpha_2 \cdot x^{k-2} + \dots + \alpha_{k-1} \cdot x + \alpha_k) \pmod p$, p — простое. Функции в семействе различаются по α_i .

Вопрос 43

Асимптотика для линейного пробирования.

- 2 – *independent* – $O(\sqrt{n})$;
- 3 – *independent* – $O(\log n)$;

- 5 – *independent* – $O(1)$;

Вообще говоря, в реальной жизни достаточно 2 – *independent* из-за энтропии входных данных.

Асимптотика для двойного хеширования.

- Если $h_1(x) \neq h_2(x)$ – 2 – *independent*, то $O(1)$;

Вопрос 44

Хеширование кукушки (описание, асимптотики) Заведем две хеш-функции h_1 и h_2 , и два массива $arr1, arr2$ с одинаковым количеством ячеек. Опишем операции:

- **Add**

1. Положим наш элемент x в первый массив в $arr1[h_1(x)]$.
2. Если ячейка была занята, вытесним из нее старый элемент y и положим его во второй массив в $arr2[h_2(y)]$.
3. Если при добавлении элемента во второй массив какой-то элемент z был вытеснен, положим его в первый массив в $arr1[h_1(z)]$.
4. Будем повторять эти действия до тех пор пока не уложим все элементы или не попадем в бесконечный цикл.
5. Если мы попали в цикл произведем `rehash`.

- **Remove**

1. Если в $arr1[h_1(x)]$ или $arr2[h_2(x)]$ есть искомый элемент, то мы его удаляем.

- **Find**

1. Если в $arr1[h_1(x)]$ или $arr2[h_2(x)]$ есть искомый элемент, то он есть во множестве.

Асимптотика:

- Мат. ожидание вставки – $O(1)$;
- На M вставок время работы – $O(1)$ с вероятностью $1 - O(\frac{1}{M})$

Вопрос 45

Разрешаем *false-positive* ошибки с вероятностью ε . Запрещаем *false-negative* ошибки.

Фильтр Блума – битовый массив из m бит и k различных хеш функций h_1, \dots, h_k , равновероятно отображающих элементы исходного множества во множество $\{0, 1, \dots, m - 1\}$, соответствующее номерам битов в массиве. Изначально, когда структура данных хранит пустое множество, все m бит обнулены. Операции:

- **Добавление элемента e :** записать единицы на каждую из позиций $h_1(e), \dots, h_k(e)$ битового массива.
- **Проверка наличия элемента e :** проверить состояние битов $h_1(e), \dots, h_k(e)$.
Если хотя бы один из них равен нулю, элемент не принадлежит множеству. Если все они равны единице, то структура данных сообщает, что элемент принадлежит множеству. При этом может возникнуть две ситуации: либо элемент действительно принадлежит к множеству, либо все эти биты оказались установлены при добавлении других элементов, что и является источником ложных срабатываний в этой структуре данных.

Оценка False Positive Rate для фиксированных k, n и m

Вероятность того, что первая функция проставит бит — $\frac{1}{m}$.

Значит вероятность того, что не проставит — $1 - \frac{1}{m}$.

Вероятность того, что конкретный бит не проставлен, после выполнения k хеш – функций — $(1 - \frac{1}{m})^k$.

Вероятность того, что конкретный бит не проставлен, после выполнения k хеш – функций и n вставок — $(1 - \frac{1}{m})^{k \cdot n}$.

Вероятность того, что конкретный бит проставлен, после выполнения k хеш – функций и n вставок — $1 - (1 - \frac{1}{m})^{k \cdot n}$.

Блум сказал, что вероятность того, что k бит проставлены, после выполнения k хеш – функций и n вставок — $\left(1 - (1 - \frac{1}{m})^{k \cdot n}\right)^k$.

Вообще говоря, эта оценка неверна из – за последнего перехода в силу отсутствия независимости битов, в 2008 году было доказано, что настоящая асимптотика:

$$\frac{1}{m^{k(n+1)}} \cdot \sum_{i=1}^m i^k \cdot C_m^i \cdot F(kn, i),$$

где $F(kn, i)$ — число сюръекций из множества размера kn в множество размера i .

Вопрос 46

Фильтр кукушки (описание).

Идея похожа на саму хеш – таблицу кукушки, но хранить мы будем не ключи, а $fingerprint : f(key)$ — хеш от ключа. Также возьмём такие хеш функции:

- $h_1(x) = hash(x)$;
- $h_2(x) = h_1(x) \oplus hash(f(x))$ (побитово)

Тогда имея $h_2(x)$ и $f(x)$ можно получить $h_1(x)$, наоборот, имея $h_1(x)$ и $f(x)$ можно получить $h_2(x)$. Далее функции реализуем по аналогии хешированием кукушки (вопрос 44), но вместо значения записываем $fingerprint$.

Вообще говоря, есть проблема: пространство $f(x)$ сильно уже пространства x , но на практике проблемы нет. Число бит на один объект:

$$\frac{\log_2 \left(\frac{1}{\varepsilon}\right) + 3}{\alpha \left(= \frac{n}{m}\right)}$$

Авторы статьи берут ($\alpha = 95, 5\% / 100$).

Вопрос 47

Задача.

Есть поток объектов, требуется посчитать сколько примерно раз объект встречается в нём.

Count-min sketch (описание).

Заводим массив $count$ и функцию h из универсального семейства. Операции:

1. $increment(x)$: произвести $count[h(x)]++$.
2. $estimate(x)$: вернуть $count[h(x)]$.

Для улучшения погрешности можно завести d копий вышеописанной структуры с разными хеш – функциями с операциями:

1. $increment(x)$: вызвать $increment(x)$ у каждой копии.
2. $estimate(x)$: вернуть минимум среди вызовов $estimate(x)$ у каждой копии.

Доказательство — 0.2.19

Вопрос 48

Задача.

Посчитать количество различных объектов во входном потоке.

HyperLogLog (описание).

Берём примерно равномерную хеш – функцию h . Для каждого объекта потока считаем хеш и смотрим на количество ведущих нулей k в результате хеширования. В структуре сохраняем самое большое из встретившихся k .

$h(x) : 1\dots$	$P \approx \frac{1}{2}$
01\dots	$P \approx \frac{1}{4}$
001\dots	$P \approx \frac{1}{8}$
...	
000\dots 1_k\dots	$P \approx \frac{1}{2^k}$

Тогда считаем, что если в структуре на данный момент сохранилось k , то во входном потоке было примерно 2^k объектов.

Объединение потоков.

Для улучшения результата можно разделить входной поток на n входных потоков (первый входной объект считается из первого потока, второй — из второго, n -й — из n -го, $n + 1$ -й — снова из первого и т. д.) и завести структуру с одним и тем же хешом для каждого потока. В каждой структуре получим какое-то k_i . Тогда считаем, что уникальных объектов было примерно $(2^{-k_1} + 2^{-k_2} + \dots + 2^{-k_n})^{-1} = \frac{1}{\frac{1}{2^{k_1}} + \frac{1}{2^{k_2}} + \dots + \frac{1}{2^{k_n}}}$.

Использование первых бит хеша в качестве номера потока.

Для того, чтобы объекты, которые сильно искажают оценку, попадали всегда в один поток и тем самым не сильно влияли на общую оценку, разделение на 2^r потоков можно обеспечить, воспринимая первые r битов хеша как номер потока, а оставшуюся часть как сам хеш:

$$h(x) = \underbrace{0010}_{r} \underbrace{11010}_{h}$$

Сканировано с CamScanner

Вопрос 49

Машина Тьюринга $M = \langle \Sigma, \Gamma, \mathbb{Q}, \delta, q_{start}, q_{accept}, q_{reject} \rangle$, такая что:

- Σ — входной алфавит.
- Γ — ленточный алфавит. $\Gamma \supset \Sigma$, $\# \in \Gamma$ — пробельный символ. Γ — конечно.
- \mathbb{Q} — множество состояний.
- $\delta : \mathbb{Q} \times \Gamma \longrightarrow \mathbb{Q} \times \Gamma \times \{L, N, R\}$ — функция переходов, где L, N, R — направления куда сдвинуться, налево, никуда и направо соответственно.
- $q_{start} \in \mathbb{Q}$ — стартовое состояние.
- $q_{accept} \in \mathbb{Q}$ — принимающее состояние. Если входное слово приводит к нему, то слово лежит в языке.
- $q_{reject} \in \mathbb{Q}$ — отвергающее состояние. Если слово приводит в него, то слово не лежит в языке.

Вычисление на машине Тьюринга.

- Начальная конфигурация: бесконечная лента, заполненная пробельными символами. На каком-то куске ленты написано входное слово. Указатель стоит на клетке с первым символом входного слова. Состояние машины — q_{start} .

- Вычисление: на каждом шаге работы машины применяется функция δ . Она возвращает символ, новое состояние и сдвиг. Тогда символ под указателем меняется на полученный, сам указатель сдвигается в нужном направлении и состояние меняется соответственно. Так происходит пока состояние не станет q_{accept} или q_{reject} . Если состояние q_{accept} , возвращается YES , если q_{reject} — NO .

Машина Тьюринга M распознаёт язык $L \subset \Sigma^*$ за время $\mathbb{T}(n)$, если $\forall x \in \Sigma^*$ вычисление ответа $M(x)$ занимает время $O(\mathbb{T}(|x|))$ и $M(x) = L(x) = \begin{cases} 1, & x \in L \\ 0, & x \notin L \end{cases}$, то есть M всегда возвращает правильный ответ.

Разрешимый язык — язык, для которого \exists машина Тьюринга, которая его распознаёт.

Вопрос 50

Многоленточная машина Тьюринга M — $\langle \Sigma, \Gamma, \mathbb{Q}, \delta, k, q_{start}, q_{accept}, q_{reject} \rangle$, где k — количество лент. Поменяется лишь определение $\delta : \mathbb{Q} \times \Gamma^k \rightarrow \mathbb{Q} \times \Gamma^k \times \{L, N, R\}^k$. Такая машина принимает во внимание и изменяет сразу k позиций на лентах.

Моделирование многоленточной на одноленточной.

Если язык L распознаётся на k — ленточной машине Тьюринга M за время $\mathbb{T}(n)$, то \exists одноленточная машина Тьюринга M' , распознающая L за $O(k \cdot \mathbb{T}^2(n))$.

Вопрос 51

Класс $DTIME(\mathbb{T}(n))$ — класс языков, который распознаётся за время $\mathbb{T}(n)$ на многоленточной машине Тьюринга.

Класс $P = \bigcup_{c=1}^{\infty} DTIME(n^c)$. Примеры:

- Определение связности графов.
- Вычисление наибольшего общего делителя.
- Задача линейного программирования.
- Проверка простоты числа.

$EXP = \bigcup_{c=1}^{\infty} DTIME(2^{n^c})$. Примеры:

- Поиск клики.
- Поиск гамильтонового пути.

Вопрос 52

Недетерминированная машина Тьюринга M — $\langle \Sigma, \Gamma, \mathbb{Q}, \delta, k, q_{start}, q_{accept}, q_{reject} \rangle$, только δ теперь многозначная функция, то есть возвращается произвольное количество вариантов действий и можно выбрать любой из них.

M принимает x , если хотя бы в одной ветви вычислений достигается q_{accept} .

Наоборот, **M отвергает x ,** если любая ветвь вычислений завершается в q_{reject} .

Время работы недетерминированной машины Тьюринга на некотором входе — \max длина ветви вычислений на этом входе.

$NTIME(\mathbb{T}(n))$ — класс языков, которые распознаются недетерминированной машиной Тьюринга за время $\mathbb{T}(n)$.

Класс $NP = \bigcup_{c=1}^{\infty} NTIME(n^c)$.

Вопрос 53

Сертификатное определение класса NP .

Язык $A \in NP \Leftrightarrow \exists$ детерминированная машина Тьюринга $V(x, s)$ (принимающая на вход x и s) — верификатор, работающая за время $poly(|x|)$, такая что $\forall x : x \in A \Leftrightarrow \exists s : V(x, s) = 1$. Аргумент s называется сертификатом.

Замечание.

То есть класс NP — такие языки, принадлежность к которым легко проверить, имея “доказательство” — сертификат.

Вопрос 54

$P \subset NP \subset EXP$.

Доказательство.

$P \subset NP$, очевидно, так как недетерминированная машина Тьюринга, просто даёт больше возможностей.

$NP \subset EXP$, так как можно смоделировать все ветви вычислений полиномиальной длины за экспоненциальное время.

Вопрос 55

Сведение по Карпу (полиномиальная сводимость).

Язык A сводится к языку B полиномиально ($A \leq_p B$), если \exists полиномиально вычислимая функция $f : \forall x \in A f(x) \in B$.

Утв. Если $B \in P$, $A \leq_p B$, то $A \in P$.

Доказательство. Сводим A к B полиномиально, разрешаем за полином, итого разрешаем за полином.

Вопрос 56

$B - NP$ – **трудный**, если $\forall A \in NP A \leq_p B$.

$B - NP$ – **полный**, если $\begin{cases} B - NP - \text{трудный} \\ B \in NP \end{cases}$

Теорема Кука-Левина (формулировка).

$SAT - NP$ – полный язык.

Справочный список основных NP -полных языков:

- $SAT = \{\varphi \mid \text{пропозициональная формула } \varphi \text{ выполнима}\};$
- $CNFSAT = \{\varphi \mid \varphi \text{ – выполнимая формула в КНФ}\};$
- $3SAT = \{\varphi \mid \varphi \text{ – выполнимая формула в 3-КНФ}\};$
- $3COL = \{G \mid \chi(G) \leq 3\}, \text{ где } \chi(G) \text{ – хроматическое число } G;$
- $01LINPROG = \{(A, b) \mid \text{для системы } Ax \leq b \text{ существует решение } x, \text{ состоящее из нулей и единиц}\};$
- $INTLINPROG = \{(A, b) \mid \text{для системы } Ax \leq b \text{ существует целочисленное решение } x\};$
- $DHAMPATH = \{(G, s, t) \mid \text{в ориентированном графе } G \text{ есть гамильтонов путь из } s \text{ в } t\}$
всевозможные вариации с неориентированным графом вместо ориентированного, циклом пути, произвольными концами вместо фиксированных);
- $LPATH = \{(G, k) \mid \text{в неориентированном графе } G \text{ есть простой путь длины } k\}$
- $SUBSETSUM = \{(n_1, n_2, \dots, n_k, N) \mid \text{из набора чисел } n_1, \dots, n_k \text{ можно выбрать подмножество суммой } N\}$ (как это соотносится с задачей о рюкзаке?);
- $PARTITION = \{(n_1, n_2, \dots, n_k) \mid \exists I \subset \{1, 2, \dots, k\} : \sum_{i \in I} n_i = \sum_{j \notin I} n_j\};$

Вопрос 57

Вероятностные вычисления на машине Тьюринга.

Вероятностная машина Тьюринга M – $\langle \Sigma, \Gamma, \mathbb{Q}, \delta, k, q_{start}, q_{accept}, q_{reject} \rangle$, только δ теперь случайная функция, т.е. в каждый момент независимо случайно идём в какую-то ветку, до куда дошли, то и возвращаем (q_{accept}, q_{reject}). Перед началом есть отдельная лента, на которой написано реализация всех случайных бит. Дальше наша машина работает, как детерминированная с входными данными $M(x, r)$. Если $|r| = k(|x|)$, то

$$\mathbb{P}(M(x) == 1) = \frac{|\{r : M(x, r) = 1\}|}{2^{k(|x|)}}$$

Вопрос 58

Опр. $B \in BPP \iff \exists$ вероятностная полиномиальная машина Тьюринга:

$$\forall x \in \{0, 1\}^* \quad \begin{cases} x \in A \Rightarrow \mathbb{P}(M(x) == 1) \geq \frac{2}{3} \\ x \notin A \Rightarrow \mathbb{P}(M(x) == 0) \geq \frac{2}{3} \end{cases} \iff$$

$$\mathbb{P}(M(x) == A(x)) \geq \frac{2}{3}, \text{ где } A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases}$$

BPP — класс языков, распознаваемых машиной Тьюринга с двусторонней ошибкой.

Опр. $A \in RP \iff \exists$ вероятностная полиномиальная машина Тьюринга:

$$\forall x \in \{0, 1\}^* \quad \begin{cases} x \in A \Rightarrow \mathbb{P}(M(x) == 1) \geq \frac{1}{2} \\ x \notin A \Rightarrow \mathbb{P}(M(x) == 0) = 1 \end{cases}$$

То есть, если $x \notin A \Rightarrow M(x)$ возвращает всегда правильный ответ. Односторонняя ошибка, можем ошибаться только на словах из языка.

Зам. Если $M(x, r) = 1$, то $x \in A$.

Опр. $A \in coRP \iff \exists$ вероятностная полиномиальная машина Тьюринга:

$$\forall x \in \{0, 1\}^* \quad \begin{cases} x \in A \Rightarrow \mathbb{P}(M(x) == 1) = 1 \\ x \notin A \Rightarrow \mathbb{P}(M(x) == 0) \geq \frac{1}{2} \end{cases}$$

То есть, если $x \in A \Rightarrow M(x)$ возвращает всегда правильный ответ. Односторонняя ошибка, можем ошибаться только на словах не из языка.

Зам. Если $M(x, r) = 0$, то $x \notin A$.

Вопрос 59

Амплификация — уменьшение ошибки.

Теорема. Амплификация в RP и $coRP$.

Опр. $A \in RP_1 \iff \exists$ вероятностная полиномиальная машина Тьюринга:

$$\forall x \in \{0, 1\}^* \quad \begin{cases} x \in A \Rightarrow \mathbb{P}(M(x) == 1) \geq \frac{1}{|x|^c} \\ x \notin A \Rightarrow \mathbb{P}(M(x) == 0) = 1 \end{cases}$$

, где $c = const$.

Опр. $A \in RP_2 \iff \exists$ вероятностная полиномиальная машина Тьюринга:

$$\forall x \in \{0, 1\}^* \quad \begin{cases} x \in A \Rightarrow \mathbb{P}(M(x) == 1) \geq 1 - \frac{1}{2^{|x|^d}} \\ x \notin A \Rightarrow \mathbb{P}(M(x) == 0) = 1 \end{cases}$$

, где $d = const$.

Тогда $RP_1 = RP = RP_2$.

Доказательство.

Очевидно, что $RP_2 \subset RP \subset RP_1$. Тогда достаточно доказать, что $RP_1 \subset RP_2$.

M уверена в ответе на $\frac{1}{|x|^c}$.

$V(x) := M(x, r_1), \vee M(x, r_2) \vee \dots \vee M(x, r_k)$.

Если $x \notin A \Rightarrow M(x, r)$ всегда 0. Значит $V(x) = 0$.

Если $x \notin A$, то $M(x, r_i)$ вернёт правильный ответ с вероятностью $\geq \frac{1}{|x|^c}$.

Тогда V вернёт неправильный ответ с вероятностью $\leq \left(1 - \frac{1}{|x|^c}\right)^k$. Хотим

$\leq \left(1 - \frac{1}{|x|^c}\right)^k \leq \frac{1}{2^{|x|^d}}$.

Подойдёт $k = |x|^{c+d}$. Получим

$$\left(1 - \frac{1}{|x|^c}\right)^k \leq \left(1 - \frac{1}{|x|^c}\right)^{|x|^c} \leq \left(\left(1 - \frac{1}{|x|^c}\right)^{|x|^c}\right)^{|x|^d} \approx \frac{1}{e^{|x|^d}}$$

Также заметим, что достаточно запустить машину полином от длины x . Чтобы добиться полиномиального времени работы. Доказали амплификацию RP . То же самое работает для $coRP$.

Вопрос 60

Теорема. Амплификация в BPP .

Опр. $A \in BPP_1 \iff \exists$ вероятностная полиномиальная машина Тьюринга:

$$A \in BPP_1 \iff \mathbb{P}(M(x) == A(x)) \geq \frac{1}{2} + \frac{1}{|x|^c}$$

, где $c = const$.

Опр. $A \in BPP_2 \iff \exists$ вероятностная полиномиальная машина Тьюринга:

$$A \in BPP_2 \iff \mathbb{P}(M(x) == A(x)) \geq 1 - \frac{1}{2^{|x|^d}}$$

, где $d = const$.

Тогда $BPP_1 = BPP = BPP_2$.

Идея доказательства — k раз запустить машину из определения BPP , нужно вернуть самый популярный ответ.

Вопрос 61

Примеры языков из вероятностных классов:

1. $PRIMES = \{n \mid n \text{ — простое в двоичной записи} \in coRP\}$.

Хотим работать за полином от $\log(n)$. Если n — простое, то алгоритм точно вернёт, что " n — простое".

Если n — составное, то алгоритм может с небольшой вероятностью вернуть, что " n — простое";

2. $PIT = \{(P, Q) \mid P \text{ и } Q \text{ — арифметические выражения, которые задают равные многочлены}\}$.

Алгоритм. Оценим степени многочленов P и Q . Возьмём \mathbb{S} из Лемма Шварца-Зиппеля в 2 раза больше чем эта степень. Берём равновероятно x из \mathbb{S} и подставляем в наши P и Q . И считаем их значения, если они совпали, то их значения скорее всего равны, если нет — не равны.

Утв. $PIT \in coRP$

Есть пара (P, Q) . Возьмём $R := P - Q$. Тогда $P \equiv Q \iff R \equiv 0$. Пусть d — верхняя оценка на степень R , то есть $d \geq \deg(R)$. А $\mathbb{S} = 2 \cdot d$.

Если $R \equiv 0$, то по равновероятно выбранным $x_1, x_2, \dots, x_n \in \mathbb{S}$ $\mathbb{P}(R(x_1, x_2, \dots, x_n) == 0) = 1$.

Если же $R \neq 0$, то по равновероятно выбранным $x_1, x_2, \dots, x_n \in \mathbb{S}$ $\mathbb{P}(R(x_1, x_2, \dots, x_n) == 0) \leq \frac{1}{2}(\frac{d}{|\mathbb{S}|})$. То что и требовалось доказать.

3. *PERFECT – MATCHING* — поиск совершенного паросочетания в произвольном графе. Совершенное паросочетание покрывает все вершины.

$PERFECT-MATCHING = \{G \mid \text{в } G \text{ есть совершенное паросочетание}\}.$

Алгоритм. Введём понятие "матрицы Татта":

$$G_1: \quad \begin{array}{c} 1 \\ \diagdown x_{12} \quad \diagup x_{13} \\ 2 \quad \quad \quad 3 \\ \diagup x_{23} \end{array}$$

$$T = \begin{pmatrix} 0 & x_{12} & x_{13} \\ -x_{12} & 0 & x_{23} \\ -x_{13} & -x_{23} & 0 \end{pmatrix} \quad x_{12}, x_{13}, \\ x_{23} - \text{ неявис.}$$

$$G_2: \quad \begin{array}{c} 2 \\ \diagdown x_{12} \\ 1 \quad \quad \quad 3 \\ \diagup x_{34} \end{array}$$

$$T = \begin{pmatrix} 0 & x_{12} & 0 & 0 \\ -x_{12} & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{34} \\ 0 & 0 & -x_{34} & 0 \end{pmatrix} \quad \begin{matrix} \text{нрвм.} \\ \text{матрица} \\ \text{Татта} \end{matrix}$$

$\exists G \text{ есть сущ. паросч.} \Leftrightarrow \underbrace{\det T}_{\text{многочлен.}} \neq 0$

Критерий существования совершенного паросочетания в терминах матрицы Татта.

В G есть совершенное паросочетание (все вершины лежат в паросочетании) $\Leftrightarrow \det T \neq 0$. А неравенство многочленов нулю мы умеем проверять в классе RP .

Вопрос 62

См. пункт 3 Вопрос 61.

Доказательство — 0.2.23

Вопрос 63

Приближённый алгоритм 2-приближения для метрической задачи коммивояжёра в полном графе.

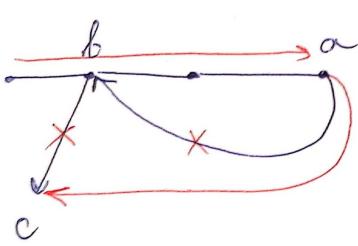
$$\forall a, b, c : \omega(a, b) + \omega(b, c) \geq \omega(a, c)$$

$$\forall a, b : \omega(a, b) \geq 0$$

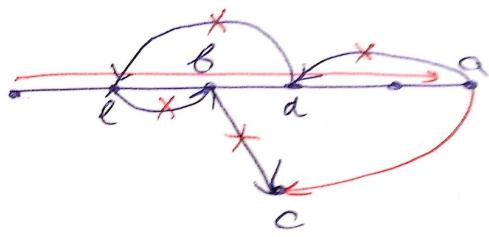
Задача: поиск цикла, посещающего все вершины **хотя бы** по одному разу, при этом \min веса.

В нашем случае задача равносильна задача поиска Гамильтона цикла, потому что:

Пусть opt — длина минимального цикла. Мы найдём гамильтонов цикл длины $\leq 2 \cdot opt$. Найдём T — минимальное оствовное дерево. $\omega(T) \leq opt$, потому что если мы возьмём минимальный Гамильтонов цикл, и выкинем из него ребро, то получим какое-то оствовное дерево. Строим по T циклический обход графа.



$$w(a, c) \leq w(a, b) + w(b, c)$$

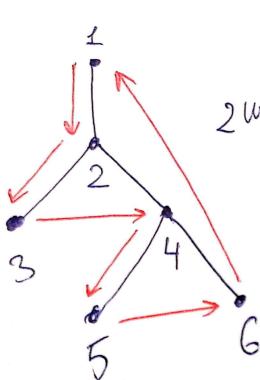


$$w(a, c) \leq w(a, d) + w(d, e) - w(e, e) + w(b, c)$$

Сканировано с CamScanner

1. Берём оставное дерево и строим по нему обход;
2. применяем алгоритм сокращения представленный в начале задачи.

Пример:



$$\rho = 12324546421$$

$$2w(T) = w(\rho) \leq 2 \cdot opt$$

Сканировано с CamScanner

Вопрос 64

Приближённый алгоритм 1.5-приближения для метрической задачи коммивояжёра

- Строим T — минимальное оставное дерево, $w(T) \leq opt$. Пусть O — множество вершин, которые в T имеют нечётную степень. $|O|$ — чётно;
- Рассмотрим подграф исходного графа, индуцированный на O (оставили все вершинки из множества O и все рёбра между ними). Найдём в этом подграфе совершенное паросочетание \min веса;
- Добавим все полученные рёбра в дерево T (могут появиться кратные рёбра). Теперь все степени чётные, этот мультиграф связен;
- Полученное дерево удовлетворяет критерию эйлеровости, поэтому мы можем построить эйлеров цикл. Строим его.
- Срезаем его до Гамильтона цикла.
- Осталось показать, что $w(\text{Гамильтонова цикла}) \leq \frac{3}{2} \cdot opt$.

Вопрос 65

Приближённый алгоритм 2-приближения для задачи о поиске вершинного покрытия.

Задача. Найти $C \subset V$, $\min |C|$, такое что $\forall e \in E$ имеет хотя бы один из концов в C .

Найдём произвольное нерасширяемое паросочетание(то есть множество рёбер M , такое что $\forall e \in E \setminus M : M \cup \{e\}$ — уже не паросочетание). Все концы всех рёбер из M — вершинное покрытие. Иначе бы было бы некоторое ребро, такое, что ни один из его концов не является концом из паросочетания, а тогда могли бы увеличить паросочетание M . Противоречие с нерасширяемостью.

Размер этого вершинного покрытия $\leq 2 \cdot opt$, где opt — размер минимального вершинного покрытия. Достаточно доказать, что $|VC| \geq |\text{паросочетание}|$. Это верно в силу того, что мы обязаны взять хотя бы одну вершинку из каждого ребра паросочетания. В частности $opt \geq |M| \implies 2 \cdot |M| \leq 2 \cdot opt$

Вопрос 66

Приближённый алгоритм $(\ln n)$ -приближения для задачи SET-COVER.

Задача. $SET - COVER = \{(n, S_1, \dots, S_k, m)\}$, где

1. n — число;
2. $S_1, \dots, S_k \subset \{1, 2, \dots, n\}$;
3. $S_1 \cup \dots \cup S_k = \{1, 2, \dots, n\}$;
4. $\exists I \subset \{1, 2, \dots, k\}, |I| = m : \cup_{i \in I} S_i = \{1, 2, \dots, n\}$

Это NP — полный язык. Задача поиска $\min m$ — NP — трудная.

Алгоритм.

Пока не всё покрыто:

1. Выбрать S_i , содержащее как можно больше непокрытых элементов;
2. Добавить S_i в коллекцию.

Доказательство — 0.2.24

Вопрос 67

Опр. Конъюнктивная нормальная форма, КНФ — нормальная форма, в которой булева функция имеет вид конъюнкции нескольких простых дизъюнктов. Пример:

$$f(x, y, z) = (x \vee y) \wedge (y \vee \neg z)$$

Приближённый алгоритм 7/8-приближения для задачи MAX-3SAT.

Задача. ϕ — форма в 3 — КНФ(в скобке ровно 3 литерала, отвечающих различным переменным $(x \vee y \vee z)$). Выполнить как можно больше скобок ϕ . m — количество скобок, $opt \leq m$, алгоритм выполняет хотя бы $\frac{7}{8} \cdot m$, значит

хотя бы $\frac{7}{8} \cdot opt$. Пусть $f(\phi(x_1, \dots, x_n))$ при фиксированных x_1, \dots, x_n — количество истинных скобок в ϕ на наборе (x_1, \dots, x_n) .

Если мы для каждой переменной будем бросать монеты и присваивать ей значение. То $\mathbb{P}(\text{скобка истинна}) = \frac{7}{8}$, так как только один ложный случай. Тогда при случайно сгенерированных x_i $E f(\phi) = \frac{7}{8} \cdot m$.

Метод условных математических ожиданий.

$$A_1^0 = E(f(\phi) \mid x_1 = 0) = E_{x_2, \dots, x_n}(f(\phi) \mid x_1 = 0).$$

$$A_1^1 = E(f(\phi) \mid x_1 = 1) = E_{x_2, \dots, x_n}(f(\phi) \mid x_1 = 1).$$

В равенстве слева, $|$ — при условии, справа — ограничение.

$$E f(\phi) = \frac{1}{2} \cdot A_1^0 + \frac{1}{2} \cdot A_1^1 = \frac{7}{8} \cdot m \implies \begin{cases} A_1^0 \geq \frac{7}{8} \cdot m \implies x_1 = 0 \\ A_1^1 \geq \frac{7}{8} \cdot m \implies x_1 = 1 \end{cases}$$

То есть та ветка, которая нас привела к мат. ожиданию $\frac{7}{8}$, мы в неё и идём. Если мы сможем построить алгоритм, который считает мат. ожидание при условии, то мы сможем совершить один шаг за адекватное время. Пусть x_1 выбрано 0. Далее считаем

$$A_2^0 = E(f(\phi) \mid x_1 = 0, x_2 = 0).$$

$$A_2^1 = E(f(\phi) \mid x_1 = 0, x_2 = 1).$$

$$E(f(\phi) \mid x_1 = 0) = \frac{1}{2} \cdot A_2^0 + \frac{1}{2} \cdot A_2^1 \geq \frac{7}{8} \cdot m \implies \begin{cases} A_2^0 \geq \frac{7}{8} \cdot m \implies x_2 = 0 \\ A_2^1 \geq \frac{7}{8} \cdot m \implies x_2 = 1 \end{cases}$$

После n итераций: $E(f(\phi)(x_1(= \alpha_1), \dots, x_n(= \alpha_n))) \geq \frac{7}{8} \cdot m$. Тут нет случайности, следовательно $f(\phi)$ — константа. Следовательно $f(\phi)(x_1(= \alpha_1), \dots, x_n(= \alpha_n)) \geq \frac{7}{8} \cdot m$.

Осталось понять, как искать $E(f(\phi) \mid x_1 = \alpha_1, \dots, x_k = \alpha_k)$. Рассмотри каждую скобку. Есть скобка $(a \vee b \vee c)$.

1. x_1, x_2, x_k не входит в эту скобку $\implies \mathbb{P} = \frac{7}{8}$;
2. Входит только одна с ложью $\implies \mathbb{P} = \frac{3}{4}$;
3. Входит две переменные с ложью $\implies \mathbb{P} = \frac{1}{2}$;
4. Входят три, все три ложные $\implies \mathbb{P} = 0$;
5. Входит хотя бы одна с истиной $\implies \mathbb{P} = 1$;

Вторая часть

Вопрос 1

Теорема Паросочетание M в двудольном графе G — max \iff в G нет увеличивающей цепи относительно M .

Доказательство

\Rightarrow :

От противного: Пусть в G с максимальным паросочетанием M существует увеличивающая цепь.

Тогда заменив в ней все рёбра, входящие в паросочетание, на невходящие

и наоборот, мы получим большее паросочетание.
То есть M не являлось максимальным. Противоречие.

\Leftarrow :

Пусть M – не max, покажем что \exists увеличивающая цепь.

Пусть M' – паросочетание: $|M'| > |M|$

Построим подграф $G' = M \oplus M'$, состоящий из ребер \in только одному из паросочетаний.

M и M' – паросочетания \Rightarrow нет вершин, которые смежны с двумя ребрами из паросочетания. То есть у каждой вершины подграфа есть не более одного ребра из M и не более одного из M' . $\Rightarrow \forall v \in G' \rightarrow \deg(v) \leq 2$

Как известно, графы с таким свойством степеней вершин представляют из себя наборы цепей и циклов.

При этом длина цикла должна быть четной, ведь иначе мы будем иметь вершину, у которой два ребра, к ней смежных, принадлежат одному паросочетанию.

В циклах поровну ребер из каждого паросочетания, значит их вклад в отрыв M' от M по числу ребер – нулевой.

Значит обогнать M у M' получится только если в графе имеется цепочка нечетной длины, у которой начальное и конечное ребра лежат в M' . Вот эта вот цепочка и есть увеличивающая.

Вопрос 2

Алгоритм Куна для поиска максимального паросочетания в двудольном графе

1. Фиксируем доли графа: L и R . Изначально считаем паросочетание пустым.
2. В доле L перебираем вершины в порядке увеличения номеров.
3. Если вершина насыщена, то пропускаем ее и идем дальше. В противном случае, пытаемся насытить вершину, запустив поиск увеличивающей цепи из этой вершины следующим образом:
 - (a) Стоя в текущей вершине v доли L , просмотрим все ребра из этой вершины.
 - (b) Возьмем текущее ребро (v, t_0) : Если t_0 – ненасыщена, то одно ребро (v, t_0) и задает нам увеличивающую цепь, просто увеличим паросочетание с его помощью и прекратим поиск. Если же t_0 насыщена каким-то ребром (t_0, p) , то пойдем вдоль этого ребра, уже в поисках увеличивающейся цепи, проходящей через ребра (v, t_0) и (t_0, p) . Для этого просто перейдем в вершину p и продолжим обход из нее.
4. В конечном итоге, мы либо найдем увеличивающую цепь из вершины v и увеличим паросочетание этой цепью, тем самым насытив вершину, либо же покажем отсутствие увеличивающей цепи и невозможность насыщения вершины.

5. Возьмем следующую по порядку вершину доли L и повторим.
6. После того как мы просмотрим все вершины, пытаясь увеличить паросочетание цепью из них, мы получим максимальное паросочетание.

Разумеется, асимптотика алгоритма будет $O(n^3)$, ведь это n применений *DFS*.

Докажем корректность алгоритма. Она следует из теоремы Бержа и того факта, что если из вершины x не существует дополняющей цепи относительно паросочетания M и паросочетание M' получается из M изменением вдоль дополняющей цепи, тогда из x не существует дополняющей цепи в M' . [Доказательство](#).

Вопрос 3

Вершинное покрытие графа $G = (V, E)$ - такое подмножество S множества вершин графа V , что любое ребро этого графа инцидентно хотя бы одной вершине из множества S .

Минимальное вершинное покрытие графа – вершинное покрытие, состоящие из *наименьшего* числа вершин.

Утверждение

Чтобы покрыть все ребра, нам нужно вершин не меньше, чем мощность наибольшего паросочетания.

Доказательство

Паросочетание это, как известно, набор непересекающихся рёбер. Мы хотим взять набор вершин, эти ребра покрывающий. Ясно что каждая вершина может покрыть не более одного ребра, ведь они не пересекаются по концам. Оценка снизу доказана.

Теорема Кенига

В двудольном графе $|M_{max}| = |C_{min}|$.

Доказательство. Явно предъявим покрытие, размер которого равен размеру максимального паросочетания.

1. Разделим граф на доли L и R .
2. Сориентируем ребра: Из $M_{max} \leftarrow$, остальные \rightarrow
3. Обойдем граф из *ненасыщенных паросочетанием* вершин $\in L$.
4. Получим разбиение графа на четыре множества: L^+, R^+ – вершины, лежащие в L или R соответственно, которые доступны из ненасыщенных вершин, лежащих в L . L^-, R^- – аналогично *недоступные* из ненасыщенных вершин, лежащих в L .
5. Поймем какие ребра бывают между каждыми из четырех множеств:

- (a) Покажем, что ребер $L^+ \rightarrow R^-$ не бывает, ведь если бы такое ребро имелось, мы из достижимой вершины, лежащей в L^+ добрались бы по этому ребру до вершины, по определению R^- , недостижимой.
- (b) Из аналогичных рассуждений понятно что не бывает ребер $R^+ \rightarrow L^-$.
- (c) Покажем отсутствие ребер $R^- \rightarrow L^+$: От противного: Пусть (r^-, l^+) – такое ребро. Это ребро вида \leftarrow , то есть принадлежащее паросочетанию.

Вершина l^+ – насыщена, значит чтобы до нее дойти мы должны были начать обход из какой-то другой вершины l' , из которой l^+ достижима (l^+ не может быть "началом" обхода, поскольку она насыщена).

Так как l' и l^+ лежать в одной доле двудольного графа, маршрут из l' в l^+ в какой-то момент должен пойти справа налево, то есть имеется ребро вида (r', l^+) , которое в силу своего направления также лежит в паросочетании. Значит из смежные ребра (r^-, l^+) и (r', l^+) лежат в одном паросочетании. Противоречие.

6. Получаем что любое ребро графа инцидентно или вершине из L^- или вершине из $R^+ \Rightarrow L^- \cup R^+$ – вершинное покрытие.

7. Покажем, что все вершины из $L^- \cup R^+$ насыщены ребрами из паросочетания:

Это так, ведь если бы были ненасыщенные вершины в L^- , то мы бы запускали из них обход, и они автоматически стали бы L^+ (так как вершина достижима сама из себя, а обход мы запускаем из всех ненасыщенных левых вершин). В свою очередь, если бы в R^+ была ненасыщенная вершина, то существовала бы увеличивающая цепь, в этой вершине заканчивающаяся, что противоречит максимальности паросочетания.

8. Как было показано ранее, не существует ребер $R^+ \rightarrow L^-$, поэтому каждому ребру паросочетания инцидентна ровно одна вершина покрытия $L^- \cup R^+$. Тогда по приведенной выше оценке мы можем сделать вывод, что требуемое покрытие найдено.

□

Вопрос 4

Алгоритм Эдмондса-Карпа для поиска максимального потока. Та же самая идея, только теперь вместо произвольного пути будем выбирать кратчайший путь по ребрам. По сути, DFS меняется на BFS.

1. Изначально поток равен 0.
2. Пока в G_f есть минимальный путь из s в t :
 - (a) $x = \min(\text{cap}_f \text{ на этом пути}) > 0$.

(b) увеличим поток f на x и изменим остаточную сеть соответствующим образом.

Утверждение: Алгоритм будет иметь асимптотику $O(|V| \times |E|^2)$. Докажем это.

Пусть $dist(u, v)$ – минимальное число ребер между вершинами u и v .

Утверждение: Пусть поток f' получается из потока f после одной итерации алгоритма Эдмондса – Карпа.

Пусть $dist'(u, v)$ – кратчайшее расстояние между вершинами u, v в $G_{f'}$, тогда $\forall v \in V \setminus \{s, t\} \hookrightarrow dist'(s, v) \geq dist(s, v)$.

То есть с каждой итерацией расстояние до источника не убывает ни у одной из вершин.

Доказательство. От противного:

Пусть $\exists v \in V \setminus \{s, t\} : dist'(s, v) < dist(s, v)$ и при этом $dist(s, v)$ – минимальное возможное среди таких v

Пусть u – предыдущая вершина перед v на кратчайшем пути из s в v в $G_{f'}$

Тогда $dist'(s, u) = dist'(s, v) - 1$ (по определению $dist$).

Также $dist'(s, u) \geq dist(s, u)$ ведь мы выбрали минимально удаленное v .

Рассмотрим 2 случая:

$(u, v) \in E_f$, тогда:

$$dist(s, v) \leq dist(s, u) + 1 \leq dist'(s, u) + 1 \leq dist'(s, v)$$

Но при этом $dist'(s, v) < dist(s, v)$

Противоречие. $(u, v) \notin E_f$, но известно что $(u, v) \in E_{f'}$.

Появление ребра (u, v) после увеличения потока означает увеличение потока по обратному ребру (v, u) . Увеличение потока производится вдоль кратчайшего пути, а значит вершина v – предыдущая перед u на пути из s в u

Это значит что $dist(s, v) = dist(s, u) - 1 \leq dist'(s, u) - 1 = dist'(s, v) - 2$, но ведь $dist'(s, v) < dist(s, v)$. Противоречие. \square

Определение: Насыщенное – ребро, вдоль которого идет поток, равный его capacity.

Утверждение: Любое ребро сети насыщается не более $O(|V|)$ раз, то есть в алгоритме $O(|V||E|)$ итераций, то есть каждая итерация насыщает хоть 1 ребро.

Доказательство.

Рассмотрим ребро (u, v) в момент его насыщения.

Если оно насыщается, то оно лежит на кратчайшем пути от s до v (так работает наш алгоритм).

Значит $dist(s, u) + 1 = dist(s, v)$.

Теперь посмотрим, когда оно могло насытиться еще один раз:

Оно сначала должно перестать быть насыщенным, для этого нужно отменить поток вдоль (u, v) , то есть пустить поток вдоль (v, u)

Пусть $dist'$ – расстояние в момент, когда пускается поток вдоль (v, u) .

$dist'(s, v) + 1 = dist'(s, u)$ (Так как проталкивание происходит вдоль кратчайшего пути).

Со временем расстояния только увеличиваются, значит $dist'(s, u) = dist'(s, v) + 1 \geq dist(s, v) + 1 = dist(s, u) + 2$

Таким образом от момента насыщения, до момента повторного насыщения расстояние $dist(s, u)$ должно вырасти по меньшей мере на 2.

Однако все расстояния ограничены $O(|V|)$, значит и насыщений может быть только $O(|V|)$. \square

Вопрос 5

Для начала определим для каждой вершины v данной сети G длину кратчайшего из s в v пути из истока и обозначим её $d[v]$ (для этого можно воспользоваться обходом в ширину).

Теорема. Расстояние между истоком и стоком строго увеличивается после каждой фазы алгоритма, т.е. $d'[t] > d[t]$, где $d'[t]$ — значение, полученное на следующей фазе алгоритма.

Доказательство. Проведём доказательство от противного. Пусть длина кратчайшего пути из истока в сток останется неизменной после очередной фазы алгоритма. Вспомогательная сеть строится по остаточной. Из предположения следует, что в остаточной сети будут содержаться только рёбра остаточной сети перед выполнением данной фазы, либо обратные к ним. Из этого получаем, что нашёлся путь из s в t , который не содержит насыщенных рёбер и имеет ту же длину, что и кратчайший путь. Но этот путь должен был быть «заблокирован» блокирующим потоком, чего не произошло. Получили противоречие. Значит длина изменилась.

Поскольку длина кратчайшего пути из s в t не может превосходить $n - 1$, то, следовательно, алгоритм Диница совершает не более $n - 1$ фазы. Таким образом, весь алгоритм Диница может выполняться за $O(V^2E)$.

Вопрос 6

Лемма 1. Пусть G — сеть, F — maxflow в G , P — её общий потенциал, а $l = dist(s, t)$. Тогда $l \leq \frac{P}{F} + 1$.

Лемма 2. P не изменяется при пропускании потока в сети.

Доказательство первой теоремы Карзанова из этих лемм (количество итераций в алгоритме Диница — \sqrt{P}). Сделаем \sqrt{P} итераций алгоритма Диница, тогда $l \geq \sqrt{P}$. Запишем тогда утверждение из леммы 1, получим $\sqrt{P} \leq l \leq \frac{P}{F_{\text{ост}}} + 1$. $F_{\text{ост}} = F - f_1$, где $f_1 \geq \sqrt{P}$ — сколько потока протолкнулось за первые \sqrt{P} итераций, как минимум на каждой итерации

проталкивалась 1 поток.

$$\begin{aligned}\sqrt{P} - 1 &\leq \frac{P}{F - f_1} \\ f_1 \geq \sqrt{P} \implies \frac{P}{F - f_1} &\leq \frac{P}{P - \sqrt{P}} \\ \sqrt{P} - 1 \leq \frac{P}{F - f_1} &\leq \frac{P}{P - \sqrt{P}} \rightarrow (F - \sqrt{P})(\sqrt{P} - 1) \leq P \\ F \cdot \sqrt{P} - F - P + \sqrt{P} &\leq P \\ F \cdot (\sqrt{P} - 1) &\leq 2P + \sqrt{P} \\ F &\leq 2\sqrt{P}\end{aligned}$$

Доказали.

Вопрос 7

DELETED!!!!

Вопрос 8

Идея. При поиске потока минимальной стоимости методом дополнения вдоль путей минимальной стоимости нам требуется находить минимальный по стоимости поток из истока в сток. Это реализуется с помощью алгоритмов поиска кратчайшего пути в графе. Поскольку стоимость некоторых рёбер может быть отрицательной, нам приходится использовать алгоритм Форда-Беллмана. Однако гораздо эффективней было бы применить алгоритм Дейкстры для этой же цели, так как у него гораздо лучше асимптотика. Для этого нам надо перевзвесить рёбра графа.

Опр. Пусть дана транспортная сеть $G(V, E)$, где V — множество вершин графа, а E — множество рёбер. Введем в каждой вершине потенциал $p(v)$. Тогда потенциальный вес (то есть стоимость) ребра (u, v) определяется как $w_p(u, v) = w(u, v) + p(u) - p(v)$.

Заметим, что сумма потенциальных весов ребер вдоль любого пути отличается от суммы весов вдоль того же самого пути на разность между потенциалом первой и последней вершины.

Применение алгоритма Дейкстры (с потенциалами) в задаче поиска k-потока минимальной стоимости. Возьмём значения потенциалов в вершинах равными минимальному расстоянию от истока до них, а расстояния найдём с помощью алгоритма Форда-Беллмана. Таким образом, нам его придётся запустить всего один раз, а не на каждом шаге алгоритма. Однако, после добавления потока вдоль кратчайшего увеличивающего пути в сети могут появиться новые ребра, равно как и исчезнуть старые, и будет необходимо пересчитать потенциалы, чтобы они оставались корректными, то есть $p(v)$ — длина кратчайшего пути от истока в вершину v в новой сети. Научимся делать это, не запуская каждый раз Форда-Беллмана.

Вдоль кратчайшего пути в сети с корректными потенциалами не появляется ребер с отрицательным весом (однако сами потенциалы уже становятся некорректными). Но так как ребер отрицательного веса нет, то мы можем пустить алгоритм Дейкстры из s , чтобы насчитать новые потенциалы. Пусть $d_1(u, v)$ — кратчайшее расстояние, найденное алгоритмом Дейкстры, из u в v в сети с появившимися новыми ребрами, но старыми потенциалами, а $d(u, v)$ — кратчайшее расстояние в новой сети без потенциалов. Нетрудно заметить, что $d_1(s, v) = d(s, v) - p(v)$, следственно, $d(s, v) = d_1(s, v) + p(v)$. Зная настоящие расстояния от истока до каждой вершины, мы теперь можем проставить новые потенциалы. Для каждой вершины v $p(v) \leftarrow d(s, v) = d_1(s, v) + p(v)$.

Кроме того, мы также нашли новый кратчайший путь из истока в сток — а значит, на следующей итерации алгоритма мы можем пустить поток по нему и повторить все заново.

Вопрос 9

Алгоритм Кнута-Морриса-Пратта с $O(|\text{pattern}|)$ доп.памяти.

Алгоритм. Дано цепочка T и образец P . Требуется найти все позиции, начиная с которых P входит в T .

Построим строку $S = P\#T$, где $\#$ — любой символ, не входящий в алфавит P и T . Посчитаем на ней значение префикс-функции p . Благодаря разделительному символу $\#$, выполняется $\forall i : p[i] \leq |P|$. Заметим, что по определению префикс-функции при $i > |P|$ и $p[i] = |P|$ подстроки длины P , начинающиеся с позиций 0 и $i - |P| + 1$, совпадают. Соберем все такие позиции $i - |P| + 1$ строки S , вычтем из каждой позиции $|P| + 1$, это и будет ответ. Другими словами, если в какой-то позиции i выполняется условие $p[i] = |P|$, то в этой позиции начинается очередное вхождение образца в цепочку.

Время работы. Префикс-функция от строки S строится за $O(S) = O(P + T)$. Проход цикла по строке S содержит $O(T)$ итераций. Итого, время работы алгоритма оценивается как $O(P + T)$.

Оценка по памяти. Предложенная реализация имеет оценку по памяти $O(P + T)$. Оценки $O(P)$ можно добиться за счет отказа от запоминания значений префикс-функции для позиций в S , меньших $|P| + 1$ (то есть до начала цепочки T). Это возможно, так как значение префикс-функции не может превысить длину образца, благодаря разделительному символу $\#$.

Вопрос 10

Рассматриваемый алгоритм состоит из $\log_n + 1$ итераций. На k — той итерации ($k = 0 \dots \log_n$) сортируются циклические подстроки длины $2k$. На последней, \log_n -ой итерации, будут сортироваться подстроки длины $2\log_n \geq n$, что эквивалентно сортировке циклических сдвигов

. На каждой итерации будем хранить массив перестановки $p[0 \dots n - 1]$, где $p[i]$ — номер суффикса, занимающего позицию i в текущей перестановке. Также будем хранить массив классов эквивалентности $c[0 \dots n - 1]$, где $c[i]$ — класс эквивалентности, которому принадлежит префикс длины $2k$

суффикса под номером $p[i]$. При этом если префикс суффикса под номером $p[i]$ лексикографически меньше префикса суффикса под номером $p[j]$, то $c[i] < c[j]$. Если же префиксы равны, то и их классы эквивалентности одинаковы. Так как мы вставили в строку символ $\$$, то к концу алгоритма каждый суффикс будет иметь уникальный класс эквивалентности, значит, мы установим порядок суффиксов.

На нулевой итерации отсортируем циклические подстроки длины 1, т.е. первые символы строк, и разделим их на классы эквивалентности (одинаковые символы должны быть отнесены к одному классу эквивалентности). При помощи сортировки подсчетом построим массив r , содержащий номера суффиксов, отсортированных в лексикографическом порядке. По этому массиву построим массив классов эквивалентности c .

На k -ом проходе имеем массивы r и c , вычисленные на предыдущей итерации. Приведем алгоритм, выполняющий k -ый проход за $O(n)$. Поскольку итераций всего $O(\log_n)$, такой алгоритм имеет асимптотику $O(n \log_n)$.

Заметим, что циклическая под строка длины $2k$ состоит из двух подстрок длины $2k - 1$, которые мы можем сравнивать между собой за $O(1)$, используя информацию с предыдущей итерации — номера классов эквивалентности c . Таким образом, для под строки длины $2k$, начинающейся в позиции i , вся необходимая информация содержится в паре чисел $\langle c[i], c[i + 2k - 1] \rangle$. Отсортируем под строки длины $2k$ по данным парам и запишем порядок в массив r . Воспользуемся здесь приёмом, на котором основана цифровая сортировка: отсортируем пары сначала по вторым элементам, а затем по первым (устойчивой сортировкой). Однако вторые элементы уже упорядочены — этот порядок задан в массиве от предыдущей итерации. Тогда, чтобы получить порядок пар по вторым элементам, надо от каждого элемента массива r отнять $2k - 1$ (r даёт упорядочение под строк длины $2k - 1$, и при переходе к строке вдвое большей длины эти под строки становятся их вторыми половинками, поэтому от позиции второй половинки отнимается длина первой половинки).

Чтобы произвести устойчивую сортировку по первым элементам пар, воспользуемся сортировкой подсчетом, имеющей асимптотику $O(n)$.

Осталось пересчитать номера классов эквивалентности c , пройдя по новой перестановке r и сравнивая соседние элементы (как пары двух чисел).

Вопрос 11

Алгоритм Касай et al

Алгоритм работает на следующем наблюдении. Рассмотрим суффикс, для которого значение lcp равно x . Если удалить первый символ из суффикса, то значение lcp нового суффикса заведомо должно быть не меньше $x - 1$. Воспользовавшись этим наблюдением, мы можем эффективно строить lcp — массив, вычисляя значение lcp в порядке убывания длины суффикса. Для каждого суффикса мы вычисляем его значение lcp , посимвольно сравнивая этот суффикс и следующий за ним в суффиксном массиве. Теперь воспользуемся тем фактом, что нам известно значение lcp суффикса, который на один символ длиннее. Следовательно, текущее значение lcp должно быть не меньше $x - 1$, где x — предыдущее значение lcp , и нам не нужно срав-

нивать первые $x - 1$ символов суффиксов. Получается алгоритм работает за время $O(n)$, поскольку производит всего $O(n)$ сравнений.

Вопрос 12

Алгоритм Укконена.

Построение суффиксного дерева для строки s путём последовательного построения суффиксных деревьев для её префиксов. Заметим, что если имеется суффиксное дерево для строки α , то суффиксное дерево для строки αc можно получить, добавив ко всем суффиксам символ c . Суффиксы рассматриваемого на текущей итерации префикса можно разбить на три группы:

1. Листья.
2. Не листья, из которых нет перехода по c .
3. Не листья, из которых есть переход по c .

Они разбиваются на три непрерывные группы. Действительно, пусть у нас есть самый короткий лист. Тогда все большие терминальные вершины так же листья, поскольку если это не так, то по ним есть несколько переходов, а из них суффиксная ссылка ведет в наш лист. Тогда и у нашего листа есть несколько переходов. Противоречие. Аналогично можно рассмотреть самый длинный нелист, у которого есть переход по c . Заметим, что

1. При добавлении c к листу он останется листом. Значит вместо того, чтобы добавлять по одному символу к листу, можно сразу завершить его всей оставшейся строкой и более не рассматривать.
2. Если из не листа нет перехода по c , то придётся создать ветвление и тогда снова получится лист, а значит его тоже можно сразу завершить.
3. Если из не листа есть переход по c , то по нему можно просто перейти и сделать следующую вершину терминальной. В добавок, по свойству суффиксных ссылок, для всех более коротких суффиксов переход по c тоже существует.

Алгоритм.

Храним ссылку на самый длинный суффикс, не являющийся листом. Изначально $cur = root$. Для каждого символа c строки s :

1. Пока нет перехода из cur по c , производим действия, описанные в первых двух пунктах и обновляем cur , переходя по суффиксной ссылке.
2. Обновляем cur , спускаясь по c .

У всех создаваемых вершин, кроме терминальных, насчитываем суффиксную ссылку с помощью процедуры сразу при создании.

Ассимптотика.

Алгоритм линеен — $O(|s|)$.

Вопрос 13

Критерий того, что строка является *longest* в своём классе.

Строка $\alpha = \text{longest}([\alpha]) \iff \begin{cases} \alpha - \text{префикс } s \\ \exists a \neq b : a\alpha \text{ и } b\alpha - \text{подстроки } s \end{cases}$

Доказательство.

\Leftarrow

1. Если α — префикс. Если мы попытаемся расширить α символом влево, то мы потеряем вхождение этого префикса.
2. Если α расширяется двумя буквами a и b , то если мы попытаемся расширить α символом влево, то мы потеряем одно из вхождений $a\alpha$ или $b\alpha$.

\Rightarrow

Докажем от противного. Пусть α — не префикс и второе условие тоже неверно. Тогда перед α всегда есть какой-то символ, так как она не может быть префиксом, а в силу второго предположения — это один и тот же символ. Значит мы можем подлить α влево, а значит α — не *longest* в своём классе. Противоречие.

Вопрос 14

В предположении SUH среднее по всем ключам мат. ожидание *успешного* поиска или удаления $O(1 + \frac{N}{M})$.

Доказательство.

Пусть наши ключи в порядке добавления были $\{k_1, \dots, k_N\}$.

Тогда время поиска значения по ключу k_i : $T(k_i) = |\{j > i : h(k_j) = h(k_i)\}|$ — число добавленных после этого совпадающих ключей, ведь как мы помним, новые пары добавляются в начало цепочки.

На языке индикаторов: $T(k_i) = \sum_{j=i}^N I(h(k_j) = h(k_i))$. Тогда среднее мат. ожидание по всем ключам это $\frac{1}{N} \sum_{i=1}^N E[\sum_{j=i}^N I_{i,j}]$

Так как $P(h(k_i) = h(k_j)) = \frac{1}{M}$, то среднее мат ожидание равняется $\frac{1}{N} \sum_{i=1}^N [1 + \frac{N-i}{M}] = O(1 + \frac{N}{M})$.

Вопрос 15

Семейство $h_{a,b}(x) = ((ax + b) \% p) \% m$ — универсально.

Доказательство.

В силу простоты p $\forall x, y \leftrightarrow ax + b \equiv ay + b \pmod{p} \iff x \equiv y \pmod{p}$

Таким образом коллизия может возникнуть только после того как мы возьмем значение по модулю m .

Проанализируем когда при взятии по модулю бывают коллизии:

Пара $(ax + b, ay + b)$ при фиксированных x и y и варыирующихся a и b пробегает все точки квадрата без диагонали. Даю установку, вы видите рисунок. Значит все точки рано или поздно будут достигнуты.

При этом $P(h(x) = h(y)) = P((ax+b)\%m = (ay+b)\%m) = P(u\%m = v\%m)$

Кроме того, $u, v \in Z_p : u \neq v$, всего таких пар $p(p-1)$.

Тогда разница между u и склеивающимися с ним значениями — ровно m .

Тогда склеивающиеся с u можно количественно оценить сверху как $(\lceil \frac{p}{M} \rceil - 1)$.

Тогда общее число склеивающихся пар оценивается как $p \cdot (\lceil \frac{p}{M} \rceil - 1)$.

Значит вероятность совпадения можно оценить как $\frac{p \cdot (\lceil \frac{p}{M} \rceil - 1)}{p(p-1)} \leq \frac{\frac{p+M-1}{M} - 1}{p-1} = \frac{1}{M}$.

Вопрос 16

Доказательство оценки матожидания времени построения.

Сначала доказываем, что матожидание количества коллизий у n_i элементов на n_i^2 бакетов меньше $1/2$. После этого применяем неравенство Маркова, чтобы найти вероятность этого. Дальше делаем то же самое с матожиданием сумм квадратов длин бакетов. После этого опять применяем неравенство Маркова. Дальше применяем лемму к этим величинам. **Лемма.** Пусть ξ_1, ξ_2, \dots — независимые случайные величины, которые принимают $\{0, 1\}$ и $P(\xi_i = 0) < p \in (0, 1)$.

Пусть $\eta = \min_k \xi_k = 1$.

Тогда $E\eta \leq \frac{1}{(1-p)^2}$.

Доказательство.

$$P(\eta = \infty) = P(\xi_1 = 0) \cdot P(\xi_2 = 0) \cdot \dots \leq \lim_{k \rightarrow \infty} p^k = 0$$

$$E\eta = \sum_{k=0}^{\infty} k \cdot P(\xi_1 = \xi_2 = \dots = \xi_{k-1} = 0, \xi_k = 1) \leq \sum_{k=0}^{\infty} k \cdot p^{k-1} = \frac{1}{(1-p)^2}$$

Тогда мы можем сделать вывод, что мат. ожидание времени перебора каждой из хеш-функций это константа. Всех этих функций $O(N)$, значит и время построение $O(N)$.

Вопрос 17

Доказательство

$h(x) = \alpha_1 \cdot x^{k-1} + \alpha_2 \cdot x^{k-2} + \dots + \alpha_{k-1} \cdot x + \alpha_k$. Параметры: $\{\alpha_i\}$. Хоти

доказать, что

$$\mathbb{P}(h(x_1) = a_1 \wedge \dots \wedge h(x_k) = a_k) = \prod_{i=1}^k \mathbb{P}(h(x_i) = a_i)$$

$$\left\{ \begin{array}{l} \alpha_1 \cdot x_1^{k-1} + \dots + \alpha_{k-1} \cdot x_1 + \alpha_k = a_1 \\ \dots \\ \dots \\ \alpha_1 \cdot x_k^{k-1} + \dots + \alpha_{k-1} \cdot x_k + \alpha_k = a_1 \end{array} \right. \iff \left(\begin{array}{ccccc} x_1^{k-1} & x_1^{k-2} & \dots & x_1 & 1 \\ x_2^{k-1} & x_2^{k-2} & \dots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_k^{k-1} & x_k^{k-2} & \dots & x_k & 1 \end{array} \right) \cdot \left(\begin{array}{c} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_k \end{array} \right) =$$

Матрица с x — матрица Вандермонда. Её определитель $\det A = \prod_{i < j} (x_i - x_j)$.

$\forall i : (x_i - x_j) \neq 0$. Определитель не равен 0 \implies решение только одно. Всего x -ш — функций: p^k . Подходит одна. Тогда $\mathbb{P}(h(x_1) = a_1 \wedge \dots \wedge h(x_k) = a_k) = \frac{1}{p^k}$. Теперь хотим доказать, что $\mathbb{P}(h(x_i) = a_i) = \frac{1}{p}$. Тогда мы докажем то, что хотели.

$$\mathbb{P}(\alpha_{k-1} \cdot x^{k-1} + \dots + \alpha_1 \cdot x + \alpha_0 = a) = \frac{1}{p}$$

Хотим доказать, что распределение равномерное на многочлене. Зафиксируем $\alpha_{k-1}, \dots, \alpha_1, x \cdot (\alpha_{k-1} \cdot x^{k-2} + \dots + \alpha_1)$. По предположению индукции и из того, что x — поле, получаем, что равномерное распределение. Тогда из оставшихся нам подходит только одно α_0 из $p \implies \mathbb{P} = \frac{1}{p}$

Вопрос 18

TODO — вычисление этого мб Вернёмся к оценки Блума $\left(1 - \left(1 - \frac{1}{m}\right)^{k \cdot n}\right)^k$, тогда при фиксированных m и n , k — оптимальное — $\frac{m}{n} \cdot \ln 2$. Фиксируем n и ε , тогда $m = \frac{-n \cdot \ln \varepsilon}{(\ln 2)^2}$. Следствие: $\frac{m}{n} = \frac{-\log_2 \varepsilon}{(\ln 2)^2} \approx 1,44 \cdot \log_2 \left(\frac{1}{\varepsilon}\right)$. А $k = \frac{-\ln \varepsilon}{\ln 2} = \log_2 \left(\frac{1}{\varepsilon}\right)$

Вопрос 19

Проанализируем, насколько мы ошибаемся в одной копии структуры. Введём следующие обозначения.

- a — вектор настоящих частот, то есть сколько на самом деле встречался каждый объект a_1, \dots, a_n ;
- \bar{a} — наш ответ.

$$\bar{a}_i = a_i + \sum_{j \neq i} X_j \mid X_j = \begin{cases} 0, & h(i) \neq h(j) \\ a_i, & \text{иначе} \end{cases}$$

$$E \bar{a}_i = a_i + \sum_{j \neq i} E X_j$$

$$E X_j = a_j \cdot \mathbb{P}(h(i) == h(j)) + 0 \cdot \mathbb{P}(h(i) \neq h(j))$$

Из универсальности хеша $\mathbb{P}(h(i) == h(j)) \leq \frac{1}{m} \implies E X_j \leq a_j \cdot \frac{1}{m}$. Тогда $\bar{a}_i \leq a_i + \frac{\|a\|}{m}$.

$$E[\bar{a}_i - a_i] \leq \frac{\|a\|}{m} \xrightarrow{\text{неравенство Маркова}} \forall c > 0 \ P(\bar{a}_i - a_i > \frac{c \cdot \|a\|}{m}) \leq \frac{1}{c} \iff P(\bar{a}_i > a_i + \frac{c \cdot \|a\|}{m})$$

Положим $c = e$, $m = \lceil \frac{e}{\varepsilon} \rceil$.

$$P(\bar{a}_i > a_i + \varepsilon \cdot \|a\|) \leq \frac{1}{e}$$

Промежуточный итог:

- *memory* : $O(\frac{1}{\varepsilon})$ — память;
- *accuracy* : $\varepsilon \cdot \|a\|$ — отличие от правильного, на сколько ошибаемся;
- *confidence* : $1 - \frac{1}{e}$ — с какой вероятностью добиваемся отличия.

Теперь рассмотрим d копий таких структур со своей хеш – функцией. Хотим уверенность $\geq 1 - \delta$. X_i — событие: i – ая даёт нужную точность ($\varepsilon \cdot \|a\|$).

$$X = \bigcup_{i=1}^d X_i.$$

$$\mathbb{P}(X) = 1 - \mathbb{P}\left(\bigcap_{i=1}^d \overline{X}_i\right) \geq 1 - \frac{1}{e^d} \text{ — confidence}$$

Положим $d = \lceil \ln \frac{1}{\delta} \rceil$.

- *memory* : $O(\frac{1}{\varepsilon}) \cdot O(\ln \frac{1}{\delta})$ — память;
- *accuracy* : $\varepsilon \cdot \|a\|$ — отличие от правильного, на сколько ошибаемся;
- *confidence* : $1 - \delta$ — с какой вероятностью добиваемся отличия.

Вопрос 20

Доказательство.

Докажем, что SAT — лежит в классе NP . Для этого предоставим сертификат — тот самый набор, который выполняет ϕ .

Теперь докажем, что $SAT - NP$ — трудный. (Дальше спасибо Адаму, мне тоже западло расписать полностью корректность, ссылка на исходник — <https://github.com/apselon/AlgorithmsAbstracts>)

Пусть $L \in NP$ то есть имеется детерм. машина $V(x, s)$, работающая за $\text{poly}(|x|)$, такая что $\forall x \rightarrow x \in L \iff \exists s : V(x, s) = 1$.

Тогда для заданного s запишем всю цепочку вычислений машины до того как она придет в q_a .

На каждом такте головка перемещается не более чем на одну клетку, а значит если машина работает за полиномиальное время, то и перемещение головки также будет полиномиальным. То есть нашу рассматриваемую область можно ограничить на $\text{poly}(x)$ до начала входных данных и на $\text{poly}(x)$

после начала входа.

Будем рассматривать таблицу, каждая строка которой — рассматриваемая часть ленты на некотором шаге. Ее высота также $\text{poly}(x)$
Заведем булеву функцию ψ , которая будет кодировать корректность вычисления, а именно то, что:

- Стартовая конфигурация именно та, которая нужна: (q_{start}, x, s)
- Каждая следующая строка получается из предыдущей с помощью перехода по δ .
- В последней строке есть q_a .

Каждая из ячеек таблицы будет переменной ψ . При этом можно утверждать, что вся таблица кроме некоторой фиксированной части, которая строится по входным данным s , фиксирована, тогда:

$x \in L \iff \exists s : \psi(s) = 1$. То есть x лежит в языке если вычисление можно сделать корректным, подставив некоторый сертификат.

Теперь будем кодировать все символы из исходного алфавита Σ последовательностями битов фиксированной длины, например 10. Теперь ячейка машины Тьюринга будет соответствовать 10 булевым переменным.

<...> Я вижу тут долгое рассуждение доказывающее корректность, а вы?
Таким образом, SAT — NP-трудный язык, а значит и NP-полный.

Вопрос 21

Теорема. Неравенство Чернова(Неравенство больших уклонений).

Есть N экспериментов: X_1, \dots, X_n .

$X_i \sim Bern(p)$ (имеет распределение Бернулли), то есть

$$X_i = \begin{cases} 1, & \text{вероятность } p \\ 0, & \text{вероятность } 1 - p \end{cases}$$

$\delta > 0$ - константа. Пусть $\bar{X} = \frac{X_1 + X_2 + \dots + X_n}{n}$.

Тогда

$$\mathbb{P}(|\bar{X} - p| > \delta p) \leq 2 \cdot \exp\left(-\frac{\delta^2 \cdot p \cdot n}{1 + \frac{\delta}{3}}\right)$$

Вопрос 22

Лемма Шварца-Зиппеля.

Пусть $P(x_1, x_2, \dots, x_n)$ — многочлен степени $d \geq 1$. $\mathbb{S} \subset \mathbb{Z}$. Если $x_1 \sim \mathbb{U}(\mathbb{S})$, $x_2 \sim \mathbb{U}(\mathbb{S})$, ..., $x_n \sim \mathbb{U}(\mathbb{S})$ (x_i выбирается равновероятно на множестве \mathbb{S}), тогда

$$\mathbb{P}(P(x_1, x_2, \dots, x_n) = 0) \leq \frac{d}{|\mathbb{S}|}.$$

Вопрос 23

Доказательство.

⇐

Рассмотрим произвольный граф G . T — его матрица Тата. Как устроен $\det T$?

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \implies \det A = \sum_{\sigma \in S_n} (-1)^\sigma \cdot a_{1,\sigma(1)} a_{2,\sigma(2)} \cdots a_{n,\sigma(n)}$$

Посчитаем $\det T$, в нашей матрицы встречаются только переменные, минус переменные и нулевые слагаемые. Тогда ненулевые слагаемые будут соответствовать только таким σ , что пара $\forall i (i, \sigma(i))$ — ребро. Оставим только те σ , которые вносят ненулевые слагаемые, то есть такие σ , что все пары, которые они задают — рёбра нашего графа. Назовём это множество L . σ — это всегда разбиение на циклы. Ненулевые слагаемые — те циклы, в которых есть все рёбра.

Если в σ есть нечётный цикл, то можно рассмотреть перестановку τ , полученную из σ обращением этого цикла. Тогда τ и σ сократятся. В чётных циклах ничего не поменяется, в нечётных же при изменении направления каждого ребра изменился знак у каждой переменной, но переменных нечётное количество, значит знак просто изменился. Также легко заметить, что знаки перестановок одинаковы. Значит, если σ вносила $+$..., то τ внесёт $-$ А значит, они сократятся.

Получается, что в $\det T$ вносят вклад только перестановки, отвечающие разбиению на циклы чётной длины. Осталось понять, что если есть разбиение на чётные циклы, то есть совершенное паросочетание. просто возьмём каждое второе ребро в цикле и получим совершенное паросочетание.

⇒

Если у нас совершенное паросочетание. То по нему построим перестановку, которая одну вершину пары переводит в другую и наоборот. Осталось заметить, что вклад этой перестановки не обнульется. Такая перестановка будет иметь вид:

$$(-x_{12}^2) \cdot (-x_{23}^2) \cdot \dots$$

Очевидно, что единственный вариант набрать такое произведение только от совершенного паросочетания.

Вопрос 24

Доказательство.

Пусть $opt = \min$ количество множеств, которое надо взять, чтобы всё покрыть. Наша цель: доказать, что цикл работает $\leq opt \cdot \log n$ итераций.

Пусть n_{curr} — количество непокрытых после какой-то итерации элементов.

Новое множество покрывает $\geq \frac{n_{curr}}{opt}$ новых элементов. Почему? Наши n_{curr} элементов можно покрыть opt множество, тогда хотя бы одно входящее в эти opt множества содержит $\geq \frac{n_{curr}}{opt}$ новых элементов. Значит

$$n_{next} \leq n_{curr} - \frac{n_{curr}}{opt} = n_{curr} \cdot \left(1 - \frac{1}{opt}\right)$$

Это означает, что на каждом шаге мы уменьшаем количество элементов хотя бы в $1 - \frac{1}{opt}$. Тогда чтобы добиться $n_{curr} = 0(n_{curr} < 1)$ нужно сделать логарифмическое количество шагов. Докажем это:

Пусть цикл делает t итераций. Хотим

$$\begin{aligned} n_{curr} \cdot \left(1 - \frac{1}{opt}\right)^t &< 1 \\ \left(1 - \frac{1}{opt}\right)^{opt} \leq \frac{1}{e} \implies n_{curr} \cdot \left(1 - \frac{1}{opt}\right)^t &\leq n_{curr} \cdot \left(\frac{1}{e}\right)^{\frac{t}{opt}} \\ \frac{t}{opt} > \ln n_{curr} \implies t > opt \cdot \ln n_{curr} &= opt \cdot \ln n \end{aligned}$$

Значит алгоритм найдет $SET-COVER$ размера $\leq opt \cdot \ln n + 1$.