Алгоритмы и модели вычислений. Контрольная 1. 27 марта 2023 Решения и критерии

Критерии написаны **в долях от единицы**.
Критерии проверки не обсуждаются. Существенные ошибки в авторском решении являются поводом к апелляции.
Апелляции принимаются только в письменном виде, дополнительные устные пояснения по работе не принимаются к апелляции. Однако вполне допустимо устное обсуждение задач и результатов с преподавателями в том числе и до подачи апелляции.
Принимаются апелляции вида “проверяющий написал, что решения нет, но вот оно – на пятой странице”, “по критериям за пункт два мне должны были поставить 0.3, а стоит только 0.2”, “корректность процедуры показана на второй строке снизу”, “контрпример проверяющего к моей конструкции не работает” и тому подобные. Не принимаются апелляции вида “я имел в виду, что вот тут граф нужно читать как матрицу, а числа как буквы”, “здесь не дописано, но далее алгоритм работает аналогично” и тому подобные.
Мы поощряем желание студента разобраться в предмете, задачах и решениях, так что корректная апелляция вида “мне завысили оценку – моё решение на самом деле неверно и по критериям я должен получить меньше” будет поощряться выгодным для студента образом.

1.♣

Верно. Для него существует разрешающая ДМТ.

Пусть S – битовый массив из $2^{2023} - 1$ бита (это количество слов длины < 2023) Пусть машина M_S устроена так: на входе w она проверяет длину слова, если длина $\geqslant 2023$, она запускает на этом слове машину M, если же длина слова меньше, она выдаёт тот бит, что находится в S на месте, соответствующем номеру слова в лексикографическом порядке.

Очевидно, для разных S (которых $2^{2^{2023}-1}$) машина M_S работает по-разному на коротких словах, но для конкретного L существует такой S, который корректно описывает принадлежность коротких слов языку L. Машина именно с этим S корректно распознаёт язык L.

Время работы M_S – константа на заполнение памяти массивом S, линейное время на проверку длины, константа для коротких слов, не более полинома от $|w|^{38}$ для длинных. Всего, полином от длины входа.

Возможны и другие решения, умещающие таблицы в состояния МТ и т.п.

1.♢

Верно. Для него существует разрешающая ДМТ.

Пусть $s \in \{0,1\}$, пусть машина M_s устроена так: на входе w она проверяет длину слова, если $w = \varepsilon$, она отвечает s. Если $w \neq \varepsilon$, то она за полином от длины w проверяет, является ли w палиндромом. После этого она запускает M на w. Если ответ на вопрос "является ли w палиндромом" есть a и $M(w) = b$, то машина M_s отвечает $a \oplus b$. Это корректно разрешает слово w, если оно не пустая строка.

В самом деле, если слово палиндром, то M отвечает неверно, так что xor даст правильный ответ, если слово не палиндром, то M отвечает верно, и M_s повторит ответ M.

Машины M_1 и M_0 выдают разные ответы на пустой строке, но одна из этих машин отвечает корректно для данного фиксированного языка L.

Время работы M_s – константа на обработку пустого слова, полиномиальное время на проверку палиндрома, не более полинома от $|w|^3$ для моделирования M. Всего, полином от длины входа.

Критерии:

1.♣

+0,1 Верный ответ

+0,3 Обоснование полиномиальности подмножества слов языка L, на которых M не останавливается

+0,3 Обоснование полиномиальности подмножества слов языка L, на которых M останавливается

+0,3 Построение разрешающей МТ языка L на основе МТ для указанных подмножеств языка

-0,05 Небольшие недочеты: отсутствие ссылки на очевидный факт, нарушение причинно-следственных связей при попытке обоснования очевидных фактов и т.д.

1.♢

+0,1 Верный ответ

+0,2 Рассмотрение случая пустого слова

+0,3 Полиномиальность языка палиндромов

+0,4 Построение разрешающей МТ языка L на основе МТ для указанных подмножеств языка

-0,05 Небольшие недочеты: отсутствие ссылки на очевидный факт, нарушение причинно-следственных связей при попытке обоснования очевидных фактов и т.д.

2.♣

Язык L принадлежит классу $\mathcal{NP}$, поскольку можно предъявить два числа – основание и показатель, предъявить сертификат простоты для каждого из этих чисел, проверка делается за полином от длины входа. Точнее, проверка простоты каждого из чисел делается за полином, вычисление факта "$p_1^{p_2}$ равно исходному числу" делается за $O(\log p_2 \cdot (p_2 \log p_1)^2)$. Это полином от длины входа $p_2 \log p_1$. Поскольку $L \in \mathcal{NP}$, а SAT $\in \mathcal{NPC}$, сводимость существует.

2.♢

Язык L принадлежит классу $\mathcal{NP}$, поскольку можно предъявить сертификат – разложение на простые делители $p+1$, сертификат простоты каждого их них, и сертификат для простоты их числа, проверка делается за полином от длины входа. Точнее, проверка простоты каждого из чисел делается за полином, различных простых делителей не больше $\log(p+1)$, так что все числа полиномиальны от длины входа $\log p$. Поскольку $L \in \mathcal{NP}$, а CLIQUE $\in \mathcal{NPC}$, сводимость существует.

Для проверки простоты можно было использовать AKS, а не сертификат

Критерии:

0,25 Верное решение, нет никаких обоснований корректности.

0,5 Верное решение, доказана корректность, но нет никакой оценки времени работы.

0,75 Верное решение, есть оценка на работу верификатора, но только от длины сертификата.

3.♣

1) Сертификат – сертификат принадлежности языку A. Проверяющий алгоритм сперва проверяет принадлежность A с помощью сертификата, затем проверяет принадлежность B за полином. По результату выдаёт ответ.

2) Построим сводимость $A \leqslant_p A \setminus B$. Поскольку $\mathcal{P} \neq \mathcal{NP}$, то $A \neq B$ и существует слово $z \in A \setminus B$. Функция сводимости

$$f(x) = \begin{cases} z, & \text{если } x \in B, \\ x, & \text{если } x \notin B \end{cases}$$

Корректность. Есть три случая:

– $x \in B$, тогда $x \in A$, тогда $f(x) = z \in A \setminus B$,

– $x \in A \setminus B$, тогда $x \in A$ и $x \notin B$, тогда $f(x) = x \in A \setminus B$,

– $x \notin A$, тогда $x \notin B$ и тогда $f(x) = x \notin A$.

Во всех случаях выполняется определение сводимости, сводимость, очевидно, полиномиальна.

3.♢

1) Сертификат – сертификат принадлежности языку A. Проверяющий алгоритм сперва проверяет принадлежность A с помощью сертификата, затем проверяет принадлежность B за полином. По результату выдаёт ответ.

2) Построим сводимость $A \leqslant_p A \cup B$. Поскольку $\mathcal{P} \neq \mathcal{NP}$, то $A \neq \overline{B}$ и существует слово $z \notin A \cup B$. Функция сводимости

$$f(x) = \begin{cases} z, & \text{если } x \in B, \\ x, & \text{если } x \notin B \end{cases}$$

Корректность. Есть три случая:

– $x \in B$, тогда $x \notin A$, тогда $f(x) = z \notin A \cup B$,

– $x \in A$, тогда $x \notin B$ и $x \in A \cup B$, тогда $f(x) = x \in A \cup B$,

– $x \notin A \cup B$, тогда $x \notin B$ и тогда $f(x) = x \notin A \cup B$.

Во всех случаях выполняется определение сводимости, сводимость, очевидно, полиномиальна.

Критерии:

0 Утверждения вида «$\mathcal{P}$ является подмножеством $\mathcal{NPC}$»

0,33 Доказана принадлежность $\mathcal{NP}$

0,67 Доказана сводимость

0,33 При доказательстве сводимости построена функция

Некритические ошибки/некритическая неполнота доказательства – потеря не больше 0,1 за каждый случай.

Не доказано $\mathcal{NP}/\mathcal{NP}$-hard означает либо отсутствие доказательства, либо критические ошибки в доказательстве, либо невозможность понять представленное доказательство за разумное время. Конкретные претензии указаны в работе.

4.
В обоих вариантах используется «фиксация» значений переменных. По сути, $\varphi|_{x_i=\alpha}$ — это новая формула, в которой на одну переменную меньше (нет переменной x_i), и которая выражает функцию, соответствующую φ при заданном значении $x_i = \alpha$.
Построение такой формулы требует в случае задачи 4.♣ линейного прохождения по формуле с заменой x_i на α (либо тождественно истинной или ложной формулы от других переменных, если не допускается иметь в формуле константы); в случае задачи 4.◇ немного сложнее: итоговая формула должна остаться КНФ, поэтому нужно соответствующим образом обработать вхождения (например, $(x_j \vee x_i)|_{x_i=0} = x_j$, а если в дизъюнкт подставляется значение литерала 1, то весь дизъюнкт равен 1, и его надо убрать из КНФ (если он не единственный). Это также можно сделать за один проход по формуле. Поэтому фиксация переменной это полиномиальное действие.
4.♣
1. Покажем, что SAT $\leqslant_p$ SAT±. Рассмотрим сводящую функцию $f : \varphi \mapsto \varphi \wedge x^*$, где x^* — новая переменная, не встречающаяся в φ. Эта функция вычислима за полином, так как она приписывает к формуле слово константной длины. Покажем корректность сводимости:
Пусть $\varphi \in$ SAT, то есть существует выполняющий набор $\vec{x}$. Тогда $f(\varphi)$ на наборе $(\vec{x}, 1)$ будет равна $\varphi(\vec{x}) = 1$, а на наборе $(\vec{x}, 0)$ будет равна 0. Значит, $f(\varphi) \in$ SAT±.
Пусть $\varphi \notin$ SAT. Значит, на любом наборе значений $\bar{x}$ будет $\varphi(\bar{x}) = 0$. Тогда и $f(\varphi)$ на любом наборе будет ложна. Значит, $f(\varphi) \notin$ SAT±.

2. Запустим оракул на φ. Если он дал отрицательный ответ, то искомой пары наборов нет, и алгоритм завершается с отрицательным ответом.
Иначе, пара наборов есть. Будем действовать так, начиная с $i = 1$: найдем $\varphi|_{x_i=1}$ и $\varphi|_{x_i=0}$, спросим оракул, лежат ли они в SAT±. Если хотя бы одна формула лежит (пусть $\varphi|_{x_i=\alpha}$), это значит, что для φ существует выполняющий и невыполняющий наборы, в которых $x_i = \alpha$. Тогда запишем $x_i = \alpha$ в оба набора и перейдем к $i \mapsto i + 1$, $\varphi \mapsto \varphi|_{x_i=\alpha}$. Заметим, что при этом сохраняется $\varphi \in$ SAT±.
Если же для какого-то i вышло так, что $\varphi|_{x_i=1}$ и $\varphi|_{x_i=0}$ не лежат в SAT±, то это значит следующее. Если формула не лежит в SAT±, то она либо тавтология, либо невыполнима. Если $\varphi|_{x_i=1}$ и $\varphi|_{x_i=0}$ обе тавтологичны, то это значит, что значение φ не зависит ни от x_i, ни от остальных переменных, то есть φ сама тавтологична, что противоречит тому, что $\varphi \in$ SAT±. Аналогичное можно сказать для случая, когда обе формулы невыполнимы. Значит, одна из формул тавтологична, и принимает значение 1 при любых значениях переменных $x_j, j > i$, а другая — значение 0. Тогда мы можем получить искомые наборы: к одному набору добавляем $x_i = 1, x_j = 1 (\forall j > i)$, а к другому $x_i = 0, x_j = 1 (\forall j > i)$.
Случай, когда $\varphi|_{x_i=1}$ и $\varphi|_{x_i=0}$ не лежат в SAT± обязательно наступит для некоторого i, так как (пусть число переменных в φ равно n) для $i = n$ он точно наступит: при фиксации x_n в формуле не останется переменных, тогда она будет константной, и однозначно не будет лежать в SAT±.
Действия алгоритма: не более $2n$ подстановок (работают за полином) и обращений к оракулу, на этом все. Значит, алгоритм работает полиномиальное от длины φ время.

4.◇
Сразу отметим, что разнообразность двух выполняющих наборов равносильна тому, что они различны: если два набора различаются, то есть некоторый индекс i такой, что $x_i \neq y_i$. Тогда одно из этих значений равно 1, а другое 0.

1. Покажем, что CNF − SAT $\leqslant_p$ DCNF. Рассмотрим сводящую функцию $f : \varphi \mapsto \varphi \wedge (x^* \vee \neg x^*)$, где x^* — новая переменная, не встречающаяся в φ. Эта функция вычислима за полином, так как она приписывает к формуле слово константной длины. Покажем корректность сводимости. Во-первых, отметим, что итоговая формула остается КНФ. Далее:
Пусть $\varphi \in$ CNF − SAT, то есть существует выполняющий набор $\vec{x}$. Тогда $f(\varphi)$ на наборе $(\vec{x}, 1)$ будет равна $\varphi(\vec{x}) = 1$, и на наборе $(\vec{x}, 0)$ будет также равна 1. Значит, $f(\varphi) \in$ DCNF.
Пусть $\varphi \notin$ CNF − SAT. Значит, на любом наборе значений $\vec{x}$ будет $\varphi(\vec{x}) = 0$. Тогда и $f(\varphi)$ на любом наборе будет ложна. Значит, $f(\varphi) \notin$ DCNF.

2. Воспользуемся доказанной сводимостью.
Как было доказано в предыдущем пункте, $\varphi \in \text{CNF} - \text{SAT} \iff \varphi' \in \text{DCNF}$, где $\varphi' = \varphi \wedge (x^* \vee \neg x^*)$.
Тогда сделаем новый оракул M' — МТ, которая за полиномиальное время проверяет КНФ на выполнимость. Она работает так: по формуле φ строит формулу φ' (полином. время), затем вызывает

оракул M.
Теперь алгоритм. Вызовем оракул M на φ. Если ответ отрицательный, пары наборов нет, алгоритм завершается с отрицательным ответом. Иначе, ищем наборы.
Найдем набор с 1. Начиная с $i = 1$ будем делать следующее: найти $\varphi|_{x_i=1}$, выяснить у M' *выполнимость* этой формулы. Если выполнима, то запоминаем $x_i = 1$, переходим к $i \mapsto i+1$, $\varphi \mapsto \varphi|_{x_i=1}$. Иначе, $\varphi|_{x_i=0}$ должна быть выполнима (иначе φ невыполнима, что не так). Тогда запоминаем и фиксируем $x_i = 0$, переходим к $i+1$. При этом сохраняется условие, что φ выполнима.
Так как при переходах сохраняется условие выполнимости, то мы обязательно дойдем до конца и зафиксируем значения всех переменных в *выполняющем* наборе. При этом, так как мы сначала пытаемся зафиксировать $x_i = 1$, то мы соберем набор с 1, если он существует. А такой набор существует, как нам изначально сказал оракул M.
Набор с 0 находится аналогично, только сначала мы пытаемся зафиксировать $x_i = 0$, а потом $x_i = 1$. Это корректно по тем же причинам, что написано выше.
Алгоритм совершает не более $4n$ (n — число переменных) подстановок (работают за полином) и и обращений к полиномиальному оракулу M', на этом все. Значит, алгоритм работает за полиномиальное время.
Критерии:

1. $0,25$ баллов
   - 0 — Сводящая функция неверная или нет никаких обоснований корректности
   - $-0,15$ — Показано следствие только в одну сторону
   - $-0,05$ — нет никаких слов о полиномиальности сводящей функции
2. $0,75$ баллов
   - 0 баллов, если алгоритм построен на оракуле, вычисляющем *выполнимость* (без пояснения, как выполнимость можно вычислить за полином с помощью исходного оракула)
   - $-0,1$ если алгоритм не дает отрицательный ответ в случае отсутствия необходимых наборов
   - $-0,25$ — нет доказательства корректности алгоритма
   - $-0,25$ — нет оценки сложности алгоритма
   - $-0,1$ — не обоснована полиномиальность подстановки значений в формулу

5.♣ и **5.◇**
1) Пусть M_1 и M_2 – два максимальных тройкосочетания в графе G. Пусть A – множество треугольников в M_1, B – множество вершин в M_2. Построим двудольный граф на A, B, рёбра между треугольником из A и вершиной из B проведём, если вершина является вершиной треугольника.
Тогда
– каждый элемент из A инцидентен некоторому ребру, в противном случае соответствующий треугольник можно было бы добавить в M_2, что противоречит максимальности,
– каждый элемент из B инцидентен не более чем одному ребру, так как треугольники в M_1 вершинно не пересекаются.
Значит, $|A| \leqslant |B|$, то есть треугольников в M_1 не больше, чем вершин в M_2, значит $|M_1| \leqslant 3|M_2|$.
2) Поскольку любое максимальное тройкосочетание подойдёт (из пункта 1)), можно использовать простой переборный алгоритм.
(1) – снять пометки со всех вершин, $M = \varnothing$
(2) – цикл по всем вершинам u, рёбрам uv и рёбрам uw
(2.1) – если $v \neq w$, и ребро vw существует, и вершины u, v, w непомечены
(2.2) – то добавить треугольник uvw в M, пометить вершины u, v, w.
Алгоритм работает за $O(V^3)$:всего вершин на шаге (2) $|V|$, количество инцидентных ей рёбер также не превосходит $|V|$. Тогда в цикле (2) проводится не более $|V^3|$ итераций, количество действий для каждой из которых не более чем константа: 5 проверок из (2.1), 3 пометки вершин и одно добавление треугольника. Алгоритм выдаёт максимальное тройкосочетание по построению: если треугольник abc можно было бы добавить в M, то вершины a, b, c оказались непомеченными в результате работы алгоритма (помечаются только добавляемые вершины). Тогда на этапе проверки рёбер ab, ac алгоритм нашёл бы и добавил abc в M.
Критерии:
Комментарий "Нет решения" к задаче или пункту означает, что данная часть решения не найдена проверяющим в присланном файле.
a) 0,4/1
–Показано, что каждый треугольник одного максимального тройкосочетания пересекается с некоторым треугольников любого другого - 0,2.
–Завершено доказательство путем оценки числа пересечений - 0,2
Возможны другие решения, к которым не применимы критерии выше
b) 0,6/1
–Корректный алгоритм - 0,2.
Полностью описаны все шаги, необходимые для получения искомого приближения
Комментарий "отсутствует доказательство корректности" означает невыполнение обоих критериев ниже:
– Доказательство корректности алгоритма(логика) - 0,2
Показано, что найденное алгоритмом является максимальным тройкосочетанием
– Доказательство корректности алгоритма(время работы) - 0,2
Оценено время работы

6.♣
Сводимость из гамильтонова пути. На малых входах (меньше 2023 вершин) делаем полный перебор, иначе следующее.
Добавляем к входу G звезду с четырьмя лучами (вершины a, b, c, d – лучи, вершина u – центр звезды). Соединяем также вершины a и b со всеми вершинами графа G. Получившийся граф назовём G'. Построение очевидно полиномиально.
Пусть в графе G есть гамильтонов путь (v_1–v_2–…–v_n). Тогда в G' есть почти гамильтонов замкнутый маршрут ($u - c - u - d - u - a - v_1 - \ldots - v_n - d - u$).
Пусть в графе G' есть почти гамильтонов замкнутый маршрут. Поскольку он проходит по всем вершинам G', он проходит по c и по d, тогда вершина u будет встречаться в маршруте по крайней мере два раза. Тогда все остальные вершины встречаются в маршруте ровно один раз. Посмотрим на кусок маршрута (a–…–b), содержащий хотя бы одну вершину исходного графа G. Этот кусок не содержит вершину u (а также c и d), так как любой маршрут от v_i к u проходит через a или через b (второй раз). Этот кусок содержит все вершины G (так как иначе для возврата в u придётся второй раз пройти через a или b). Следовательно, в этом куске есть маршрут, посещающий все вершины графа G ровно по одному разу, т.е. гамильтонов путь.
6.♢
Сводимость из антиклики. На малых входах (меньше 2023 вершин) делаем полный перебор, иначе следующее.
Добавляем к каждой вершине входа G усик – одну соседнюю вершину с ребром. Получившийся граф назовём G'. k меняем на $n+k$, где n – число вершин в G. Построение очевидно полиномиально. Новые вершины будем помечать штрихами.
Пусть в графе G есть антиклика A размера k, тогда $A \cup V'$ почти независимое множество размера $n+k$.
Пусть в графе G' есть почти независимое множество размера $n+k$, назовём его S. Добавим в S все штрихованные вершины и уберём все нештрихованные вершины, имеющие нештрихованных соседей – полученное множество назовём S_1. Если S содержало нештрихованную вершину вместе с её штрихованной, то S_1 также содержит их обеих. В противном случае S содержало не более одной вершины из пары штрихованная-нештрихованная, а S_1 содержит как минимум одну (штрихованную). Значит $|S_1| \geqslant |S|$. Тогда $S_1 \setminus V'$ – независимое множество размера $|S_1| - n \geqslant |S| - n = k$ в графе G.
Альтернативное решение. Сводимость из антиклики.
Сделаем дубликат каждой вершины, соединённый с оригиналом: если v и u соединены в G, то в G' будут соединены попарно все четыре вершины v,v',u,u', $k \to 2k$.
Пусть в графе G есть антиклика A размера k, то в G' есть почти независимое множество размера $2k$ – это $A \cup A'$.
Пусть в графе G' есть почти независимое множество размера $2k$, назовём его S. "Спроецируем" множество S на G, получим S_0. S_0 – это набор изолированных вершин S_1 и набор непересекающихся клик размера два S_2 (иначе есть вершина степени два, которая до проекции также была вершиной степени два). Пусть две вершины в S_2 соединены ребром – тогда в исходном S ему соответствовали две вершины (а не более), то есть число вершин не изменилось. Возьмём из каждой пары в S_2 одну из вершин. S_1 и половина S_2 вместе образуют антиклику размера $|S_1| + |S_2|/2 = \dfrac{|S_2| + 2|S_1|}{2} \geqslant \dfrac{|S|}{2} = k$.
Критерии:

- 0 Предложенный к рассмотрению текст не является решением задачи
- 0 Некорректная сводимость
- 0,3 Построена правильная сводимость (или сводимость, некорректная только для конечного набора входов)
- +0,3 Для правильной сводимости доказана корректность в одну сторону ($x \in L \to f(x) \in L'$)
- +0,4 Для правильной сводимости доказана корректность сводимости в обратную сторону ($f(x) \in L' \to x \in L$)

-0,1 Небольшие ошибки в доказательствах корректности

7.♣
Предположим, что A полиномиальный, возьмём граф G. Пусть $A(G) = k$, тогда $\#MC(G)$ (это число максимальных клик) лежит в отрезке $[k/2, 2k]$.
Если $k > 8$, то $\#MC(G) > 4$. Тогда добавим в G клику размера 1, получим G_1, запустим $A(G_1)$, затем добавим в G клику размера 2, получим G_2, запустим $A(G_2)$, затем добавим в G клику размера 3, получим G_3, запустим $A(G_3)$ и так далее. Покуда i в G_i не превосходит $c(G)$, выполняется $\#MC(G_i) \geqslant \#MC(G) > 4$, тогда $A(G_i) > 2$. Как только i стало больше $c(G)$, выполняется $\#MC(G_i) = 1$, а тогда $A(G_i) \leqslant 2$.
Если $k \leqslant 8$, то $\#MC(G) \leqslant 16$. Тогда добавим в G 64 клики размера 1, получим G_1, запустим $A(G_1)$, затем добавим в G 64 клики размера 2, получим G_2, запустим $A(G_2)$, затем добавим в G 64 клики размера 3, получим G_3, запустим $A(G_3)$ и так далее. Покуда i в G_i меньше $c(G)$, выполняется $\#MC(G_i) = \#MC(G) \leqslant 16$, тогда $A(G_i) \leqslant 32$. Как только i станет равным $c(G)$, выполняется $\#MC(G_i) = 64 + \#MC(G) > 64$, тогда $A(G_i) > 32$.
Стало быть, не более чем за $|V|$ обращений к алгоритму A мы узнаем размер максимальной клики в G, что невозможно в предположении $\mathcal{P} \neq \mathcal{NP}$.
7.◊
Предположим, что A_k полиномиальный, возьмём граф G.
Возьмём $3k$ копий G, назовём их $G_1, \ldots, G_{3k}$. Объединим их в один граф H и добавим рёбра: каждая вершина копии G_i связывается ребром с каждой вершиной каждой другой копии G_j, $j \neq i$. Построение H полиномиально.
Тогда $\chi(H) = 3k\chi(G)$, потому что
– для каждой копии хватит $\chi(G)$ цветов по определению хроматического числа, в каждой копии можно использовать свои цвета, так что $\chi(H) \leqslant 3k\chi(G)$,
– в каждой копии нужно использовать "свежие" цвета, поскольку вершина каждой копии связана со всеми вершинами всех прочих копий по построению, так что $\chi(H) \geqslant 3k\chi(G)$.
Запустим $A(H)$, ответ будет лежать в отрезке $[\chi(H) - k, \chi(H) + k] = [3k\chi(G) - k, 3k\chi(G) + k]$. Тогда число $\dfrac{A(H)}{3k}$ лежит в отрезке $[\chi(G) - \frac{1}{3}, \chi(G) + \frac{1}{3}]$, его целая часть в точности равна $\chi(G)$.
Критерии:

- Текст решения отсутствует: 0
- Предоставленный текст не является решением задачи: 0
- Предложена конструкция, использующая данный алгоритм и разрешающая $\mathcal{NPC}$ задачу за полином, но её корректность не доказана: 0.5
- Предложена конструкция, использующая данный алгоритм и разрешающая $\mathcal{NPC}$ задачу за полином, её корректность не доказана, но в конструкции или доказательстве были оставлены лакуны: 0.75
- Приведено верное решение: 1