САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДОРАСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Отчётная работа

Алгоритм Δ-раскраски списочного графа

Выполнил: Подлужный Иван
Преподаватель: О.В.Алимова
20.12.2016

Санкт-Петербург 2016г.

Содержание

1 Описание алгоритма

1.1 Определения

Определение. Графом $G(E, V)$ назовём пару из конечного непустого множество E и множества вершин и $V \subset E \times E$ рёбер. Мы рассматриваем графы неориентированные графы – $\forall u, v \in E$ $(u, v) \in V \Leftrightarrow (v, u) \in V$ и без петель – $\forall u \in E$, $(u, u) \notin V$.

Определение. Пусть G – граф и $E_1 \subset E$, $E_1 \neq \varnothing$ и $V_1 \subset E_1 \times E_1 \cap V$. Тогда граф на $G(E_1, V_1)$ назовём подграфом G.

Определение. Пусть G – граф. Будем говорить, что подграф $H < G$ индуцированый, если $\forall u, v \in V(H)$ $(u, v) \in E(G) \Leftrightarrow (u, v) \in E(H)$.

Определение. Пусть $G(V, E)$ – граф. Степенью вершины $v \in E(G)$ назовём $d_G(v) = \#\{u \in E | (u, v) \in V(G)\}$.

Определение. Пусть $G(V, E)$ – граф и $H \subset V(G)$. Окрестностью H $N_G(H) \subset V(G)$, $N_G(v) = \{u \in V(G) \setminus H | \exists v \in H : (v, u)$

Определение. Граф $G(V, E)$ регулярен, если $\forall v, u \in E(G)$, $d_G(v) = d_G(u)$

Определение. Пусть $G(V, E)$ граф и $v, u \in E(G)$. Простым путём из v в u, $u \neq v$ назовём подграф $P(V_1, E_1)$ $V_1 = \{x_i\}_{i=0}^n$, $x_0 = v$, $x_n = u$, $\forall i, j \in \mathbb{N}$ $0 \leqslant i, j \leqslant n$ $x_i \neq x_j$, $E_1 = \{(x_i, x_{i+1}) \in E(G) | 0 \leqslant i < n\}$.

Определение. Пусть $G(V, E)$, $H \subset V(G)$ и $u, v \in V(G)$. Будем говорить, что путь (простой) $P(V_p, E_p)$ соединяющий v и u обходит H, если H не содержит вершин P, кроме быть может u, v.

Определение. Граф $G(V, E)$ связен, если $\forall u, v \in E(G)$ существует простой путь.

Определение. Подграф G_1 графа G является компонентой связности, если G_1 максимальный по включению связный граф G.

Определение. Граф $G(V, E)$ двусвязен, если $\forall u, v \in E(G)$ существуют два простых пути P_1 и P_2, $V(P_1) \cap V(P_2) = \{u, v\}$.

Определение. Граф $G(V, E)$ полный или, эквивалентно G – клика, если $\forall u, v \in E(G)$ $\exists (u, v) \in V(G)$. Будем его обозначать также $K_{\#V(G)}$

Определение. Пусть $G(V, E)$. Тогда $\delta G = \inf\limits_{v \in V(G)} d_G(v)$, $\Delta G = \sup\limits_{v \in V(G)} d_G(v)$

Определение. Граф $G(V, E)$ называется циклом, если G связен, регулярен и $\Delta G = 2$. Будем обозначать его также $C_{\#E(G)}$.

Определение. Пусть $a, b, c \in \mathbb{N}$. $\theta(a, b, c)$-графом назовём граф, образованный вершинами u, v и тремя попарно непересекающимися путями, содержащими a, b, c рёбер.

Определение. Пусть H – цикл. Граф G, образованный вершинами и рёбрами H, вершиной $u \notin H$ и $\forall v \in V(H)$ $\exists (u, v) \in V(G)$ назовём веретеном.

Определение. Пусть H – цикл. Граф G, образованный вершинами и рёбрами H, вершиной $u \notin H$, рёбрами, соединяющие u с H и $\#\{(u, v) \in E(G) | v \in V(H)\} \geqslant 2$ назовём зонтиком.

Определение. Граф $G(V, E)$ называется деревом, если G связен и $\nexists G_1 < G$, G_1 – цикл.

Определение. Подграф G_1 графа G блок, если G_1 максимальный по включеню двусвязный подграф G.

Определение. Пусть $G(V, E)$ граф, $H \subset V(G)$ Тогда $G \setminus \{H\}$ есть граф $Q(V_1, E_1)$, $V_1 = V \setminus H$, $E_1 = \{(u, v) \in E(G) | u, v \in V_1\}$.

Определение. Пусть $G(V, E)$ связный граф. Тогда $v \in V(G)$ точка сочленения, если $G \setminus \{v\}$ не является связным.

Определение. Пусть $G(V, E)$ связный граф. Тогда $B(G)(V_b, E_b)$ дерево блоков и точек сочленений, если $V_b = \{B_i < G | B_i - \text{блок}\} \cup \{v_i \in V | v_i - \text{точка сочленения}\}$, $V_b \subset E_b \times E_b$ и $(a, b) \in V_b$, если a блок, b точка сочленения и $b \in a$ или наоборот.

Замечание. Указанный в предущем определении граф – дерево и его листья – блоки.

Доказательство. Из построения следует, что каждая вершина в $B(G)$, отвечающая точке сочленения связана с вершинами, отвечающими блокам и наоборот.

Покажем, что $B(G)$ связен. Действительно, если $\exists u, v \in V(B(G))$, между u, v не существует пути, то рассмотрим эти точки сочленения или блоки в исходном графе G. Не умаляя общности, B_u и B_v – блоки. Если бы здесь существовал путь из $a \in B_u$, $b \in B_v$, то отмечая проходимые путём блоки и точки сочленения, получим путь и в $B(G)$. Противоречие.

Действительно, если $a_1 \dots a_n$ цикл в $B(G)$, то в нём цикл имеет хотя бы 2 вершины отвечающие блокам. Рассмотрим подграф в исходном графе, отвечающий циклу в нашем графе. Тогда при удалении любой $v \in V(G)$ подграф остаётся связным, следовательно является компонентой двусвязности и отображается на $B(G)$ одной вершиной.

Если $v \in V(B(G))$ лист, отвечающий точке сочленения, то так как она лежала только в одном блоке, удаление этой точки сочленения оставляет граф связным, следовательно она не являлась точкой сочленения. $\square$

Определение. Пусть $G(V, E)$ связный граф. Блок $B < G$ отвечающий листу в дереве $B(G)$ назовём крайним блоком.

Определение. Пусть $G(V, E)$ связный граф и $(u, v) \in E(G)$. Обозначим за $G \cdot (u, v)$ – G со стянутым ребром (u, v) граф $G \setminus \{u, v\}$, пополненный вершиной $i \in V(G \cdot (u, v))$ и рёбрами $(i, a) \in E(G \cdot (u, v))$ и $(i, b) \in E(G \cdot (u, v))$ для любых $a, b \in V(G)$, что $(u, a) \in V(G)$, $(i, b) \in V(G)$.

Определение. Пусть $G(V, E)$ граф и $H < G$. Тогда $G \cdot H$ – граф, полученный последовательным стягиванием рёбер в $E(H)$.

Замечание. Если H связен, то $V(G \cdot H) = \{i\} \cup V(G) \setminus V(H)$ и $E(G) = \begin{cases} (u, v) & u, v \in V(G) \setminus V(H) \\ (i, v) & \exists h \in V(H) | (h, v) \in E(G) \end{cases}$

Определение. Пусть $G(V, E)$ связный граф. Тогда G – граф Брукса, если $G \neq C_{2k+1}$, $G \neq K_n$, $n, k \in \mathbb{N}$.

Определение. Пусть $G(V, E)$ граф. Каждой $v \in V(G)$ сопоставим $\mathscr{P}_v \subset \mathbb{N}$, $\#\mathscr{P}_v < \infty$ – список цветов v, $\mathfrak{A} = \{\mathscr{P}_v\}_{v \in V(G)}$. Назовём пару $(G, \mathfrak{A})$ списочным графом.

Определение. Пусть $(G, \mathfrak{A})$ списочный граф. Тогда $f\ V(G) \to \mathbb{N}$ называется правильной вершинной списочной раскраской G, если

1. $\forall v \in V(G)$, $f(v) \in \mathscr{P}_v$
2. $\forall u, v \in V(G)$, $(u, v) \in E(G) \Rightarrow f(u) \neq f(v)$

Определение. Пусть $G(V, E)$ граф. Назовём его d_G-списочно раскрашиваемым, если существует $\mathfrak{A}$, $\forall v \in V(G)$, $\#\mathscr{P}_v \leqslant d_G(v)$ и существет правильная вершинная списочная раскраска $(G, \mathfrak{A})$.

Определение. Пусть $G(V, E)$ граф. Назовём его Δ-списочно раскрашиваемым, если существует $\mathfrak{A}$, $\forall v \in V(G)$, $\#\mathscr{P}_v \leqslant \Delta(G)$ и существет правильная вершинная списочная раскраска $(G, \mathfrak{A})$.

Определение. Пусть G связный граф. Остовным деревом назовём дерево $H < G$, такое что $V(H) = V(G)$.

Определение. Пусть G связный граф. Деревом поиска в глубину с корнем w назовём остовное дерево построенное по следующему алгоритму:

Пока есть ранее не отмеченный сосед v вершины u – отметить, перейти к вершине v. Если таких соседей не оказалось, добавить вершину v в дерево, перейти к той вершине u, из которой пришли, добавить ребро (u, v).

Если $v \in V(G)$, $v \neq w$, то предком v будем называть ту вершину, из которй мы пришли в v в ходе алгоритма.

Ясно, что посторенный граф является деревом. Действительно, если C – цикл в H. Ясно, что весь цикл не может состоять из предков одной вершины, так как последний потомок не мог быть соединён с двумя вершинами в цикле. Но в цикле каждая врешина связана с двумя другимим. Ребро проводилось только между потомком и предком, так что выберем вершину c, потомок которй не лежит в C. По построению, c связана только с одной вершиной в C. Противоречие.

1.2 Теоретическая часть

Основным утверждением будет следующая теорема:

Теорема 1.1. *Пусть G – граф Брукса. Тогда G Δ-списочно раскрашиваем.*

Начнём со вспомогательных лемм

Лемма 1.1. *Любой зонтик U содержит индуцированный θ граф или C_{2n} как подграф.*

Доказательство. Пусть зонтик, не являющийся веретеном, U состоит из вершин цикла c_i и центальной вершины w. Пусть $n = \#\{(w,u) \in E(U)\}$.

1. $n = 2$. Тогда $U - \theta$ граф.
2. $n = 3$. Пусть $\{c_1, c_i, c_j\}$, $1 < i < j$ – вершины соединённые с w. Так как U не веретено, то не умаляя общности $\exists\{_2 \cdots c_{i-1}\}$ и $\nexists(c_k, w) \in E(G)$. Тогда $U \setminus \{_2 \cdots c_{i-1}\}$ – индуцированный θ подграф.
3. $n \geqslant 4$. Тогда пусть $\{c_1, c_i, c_j\}$ – вершины, соединённые с w и в промежутках $\{_2 \cdots c_{i-1}\}$ и $\{_{i+1} \cdots c_{j-1}\}$ нет вершин, соединённых с w. Тогда граф индуцированный на $\{w, c_1, \ldots, c_j\}$ – индуцированный θ граф.

□

Теорема 1.2. *Любой двусвязный граф Брукса G содержит θ-граф или C_{2k} в как индуцированный подграф.*

Доказательство. По лемме 1.1, достаточно показать, что G содержит или зонтик, или θ-граф, или чётный цикл как индуцированный подграф. Рассмотрим граф G. Так как он двусвязен, $\delta(G) \geqslant 2$, значит G содержит цикл, а значит и индуцированный цикл $C < G$. Если цикл чётный, то доказательство закончено.

Пусть $C = C_3$, то есть треугольник. Дополним C до максимальной по включению клики K, содержащей C. Пусть $\{v_1, \ldots, v_k\} = V(K)$. Так как G не был кликой, то $N_G(K) \neq \varnothing$. Рассмотрим $w \in N_G(K)$. Если $\forall v \in V(K)$, $(w,v) \in E(G)$, то w вместе со всеми рёбрами и K образуют клику, что противоречит максимальности K. Значит $\exists v_i \in V(K)$, что $(w, v_i) \notin V(K)$. Если $\exists v_l, v_j \in V(K)$, $i \neq j$ $v_i, v_j \in N_G(w)$, то граф, индуцированный $\{w, v_i, v_j, v_l\}$ – $\theta(1,2,2)$ индуцирванный граф.

Если $\forall w \in N_G(K)$, $\#N_G(w) \cap V(K) = 1$, то рассмотрим v_i, смежную с w. Рассмотрим крачайший путь от w до какой-либо вершины в $v_j \neq v_i$ в K. Так как G двусвязен, то такая вершина обязательно найдётся. Пусть P такой путь. Тогда если $v_l \in V(K)$, $l \neq i$, $l \neq j$, то $\{v_i, v_j, v_l\} \cup V(P)$ индуцируют зонтик с центром v_l.

Пусть $\#V(C) > 3$. Тогда $C \neq G$, так как G – граф Брукса. Если для некоторй $w \in V(G)$ выполнено $\#N_G(w) \cap C \geqslant 3$, то можно выбрать $\{c_1, c_i, c_j\}$ так, что в промежутках $\{_2 \cdots c_{i-1}\}$ и $\{_{i+1} \cdots c_{j-1}\}$ нет вершин, соединённых с w и $\{w, c_1, \ldots, c_j\}$ – индуцированный θ граф.

Пусть $\#N_G(w) \cap C = 2$. Тогда $\{w, V(C)\}$ – индуцированный θ-граф.

Пусть $\forall w \in N_G(C)$ $\#N_G(w) \cap C = 1$, то рассмотрим $c_i \in N_G(w) \cap C$. Соединим w кратчайшим путём P до некоторой другой вершины $c_j \in C$, $c_i \neq c_j$. Тогда $V(P) \cup V(C)$ индуцируют θ-граф. □

Теорема 1.3. *Любый регулярный граф Брукса содержит индуцированный граф Брукса.*

Доказательство. Рассмотрим дерево блоков и точек сочленения исходного графа. Рассмотрим в нём крайний блок. Он будет двусвязным и точка сочленения будет иметь степень, меншую, чем любая другая точка этого блока, так как в исходном графе все степени равны, но одно хотя бы из смежных рёбер точки сочленения не лежит в нашем блоке. Значит не все степени однинаковы и блок является графом Брукса. □

Теорема 1.4. *Пусть связный граф G содержит индуцированный подграф I, являющийся d_I-списочно раскрашиваемым. Тогда G d_G-списочно раскрашивавем.*

Доказательство. Стянем I и рассмотрим $G \cdot I$. Рассмотрим остовное дерево поиска в глубину с корнем в i. Начнём красить его от листьев к корню. Так как длинна списка каждой вершины хотя бы d_G, то у кажой некорневой вершины будет хотя бы 1 свободный цвет, так как предок ещё не был раскрашен. Таким образом, мы сможем покрасить каждую вершинку $G \cdot I$, кроме быть может i. Перенесём получившуюся раскраску $V(G) \setminus I$ на G. Так как при стягивании не образовывались и не удалялись рёбра ни из $G \setminus I$, то раскраска останется корректной.

Расмотрим подграф I. Для каждой $v \in V(I)$ выкинем из списка $\mathscr{P}_v$, все цвета, в которые уже покрашены $N_G(v) \setminus I$. Таким образом $\#\mathscr{P}'_v \geqslant d_G - (d_G - d_I) = d_I$. Следовательно для набора $\mathfrak{A}_I = \{\mathscr{P}'_v\}_{v \in V(I)}$, $(I, \mathfrak{A}_I)$ существует правильнная вершинная списочная раскраска. Так как I был индуцированным то, её можно корректно объединить с раскраской G. □

Докажем важные утверждения про раскрашиваемость "элементарных"графов.

Теорема 1.5. *C_{2n} $d_{C_{2n}}$-списочно раскрашиваем.*

Доказательство. Пусть дан $(C_{2n}, \mathfrak{A})$. Если все списки одинаковы, то решение тривиально – красить последовательно. Так как число вершин чётно, то это возможно. Так что пусть смежные вершины v_1 и v_2 имеют различные списки. Тогда в списке v_1 содержится цвет k, которого нет в списке v_2. Тогда начнём красить вершины C_{2n} от v_1 в направлении, противоположном v_2. Каждый раз мы сможем красить вершины, пока не придём в v_2. Но у v_2 соседи v_1 и v_3, причём цвет первого не содержится в списке v_2. Значит у списка v_2 есть цвет, не совпадающий с цветом v_3. Его и выберем. □

Теорема 1.6. *θ-граф списочно раскрашиваем.*

Доказательство. Пусть дан $(G, \mathfrak{A})$ и u, w – концы θ-графа и пусть $1 < a \leqslant b \leqslant c$. Вершины путей P_a, P_b, P_c обозначим как $V(P_a) = \{a_1 \dots a_k\}$, $V(P_b) = \{b_1, \dots, b_j\}$, $V(P_C) = \{c_, \dots, c_l\}$. Пусть $N_G(u) = \{a_1, b_1, c_1\}$ и ни одна из вершин не равна w. Тогда все их списки длины 2 и по принципу Дирихле, в списке u есть цвет, которого нет в в списке одной из вершин c_1, не умаляя общности. Покрасим её в этот цвет. Тогда будем красить вершины в поряде $w, a_1, \dots, a_k, b_1, \dots, b_j, u, c_k, c_{k-1}, \dots, c_2$. Остановимся на c_1. Отметим, что у неё есть хотя бы 1 свободный цвет, так как у её соседа w цвет не из списка c_1. Следовательно, c_1 можн корректно покрасить. □

Следствие. Пусть G – регулярный граф Брукса. Тогда G Δ-списочно раскрашиваем.

Доказательство. По теореме 1.3 в G есть индуцированный двусвязный граф Брукса H. По теореме 1.2 из H можно бывелить один из двух выше рассматриваемых элементарных графов U. По двум вышедоказанным леммам, они d_U-списочно раскрашиваемы, значит и G d_G-списочно раскрашиваем по теореме 1.4. Следовательно, уж тем более G Δ-списочно раскрашиваем. □

Теорема 1.7. *Пусть G нерегулярный граф Брукса и $d_G(w) \neq \Delta$. Тогда G d'_G-списочно раскрашиваем, где $d'_G(v) = d_G(v)$, при $v \neq w$ и $d'_G(w) = d_G(w) + 1$.*

Доказательство. Построим остовное дерево поиска в глубину с корнем в w и начнём его красить от листев к корню. Для кажой некорневой вершины у нас будет d_G доступных цветов и не более $d_G - 1$ исползованных, так как предок вершины в дереве ещё не покрашен. Для корня у нас возможно окажутся заняты все d_G цветов. Тогда пополним наш список w ещё одним цветом и покрасим в него. □

Основная теорема является очевидным следствием предыдущей теоремы и следствия теоремы 1.6

2 Описание алгоритма

Алгоритм реализует теоретическую модель по следующей схеме:

1. Ввод графа посредством считывания матрицы смежностей и списком цветов каждой вершины.
2. Разбиение графа на компоненты связности.
3. Анализ случаев и отсечение нерегулярных и полных графов.
4. Поиск в графе компоненты двусвязности.
5. Поиск в двусвязной компоненте графа элементарных индуцированных подграфов.
6. Покраска элементарных компонент.
7. Покраска компоненты связности с вырезанным подграфом и объединение раскрасок.
8. Объединение раскрасок по различным компонентам связности.
9. Вывод итоговой раскраски вершин всего графа.

Программа реализована на MSVisual Studio 2010 на языке C++.

2.1 Ввод

Стандартный вход программы – число $n \in \mathbb{N}$, считываемое с консоли, затем $\frac{n(n-1)}{2}$ чисел, составляющих нижний треугольник матрицы смежности.

```
4

1
0
1
1
1
1
```

Затем пользователь вводит на каждую вершину список цветов – последовательность различных чисел $\{a_n\} \subset \mathbb{Z} \setminus \{0\}$, $a_0 \neq 0$ и $\exists k \in \mathbb{N}$: $a_k = 0$. Цвета – элементы последовательности до первого нулевого члена.

```
0 я вершина. Введите список цветов и закончите 0.

3
2
4
1
0
```

Длинна списка цветов должна удовлетворять задаче – если G несвязное объединение графов, то длинна списка для вершины i должна быть хотя бы Δ_{G_i}, где G_i – компонента связности G, $v_i \in G_i$ и G_i – граф Брукса и $\Delta_{G_i} + 1$ в противном случае.

2.2 Вывод

Стандартный вывод программы – полная матрица смежности и список раскраски вершин в цвета.

```
Making graph

* 0 1 2 3

0 0 1 0 1
```

```
1 1 0 1 1
2 0 1 0 1
3 1 1 1 0

 ***

0    2
1    3
2    2
3    1
Printed
```

2.3 Код программы

2.3.1 Хранение матрицы и списка цветов

Граф G хранится как элемент класса `Graph`, состоящего из матрицы смежноти `vector<vector<bool» Matrix` и списка вершин `vector<graphvert> Vert`.

Вершина хранится как элемент класса `graphvert`, состоящего из

1. `int Color` – цвета вершины, по умолчанию 0.
2. `vector<int> list` – список цветов вершины.

2.3.2 Обработка входных данных

Обработка осуществляется посредством функций `vector<graphvert> makelist(int N)`, запрашивающей у пользователя список цветов N вершин и конструктора класса `Graph(vector<graphvert> L, vector<vector<bool > > M)`, который по списку вершин и матрице смежности обрабатывает граф.

2.3.3 Выделение компонент связности графа

Эту операцию осуществляет фнкция `vector<pair< Graph , vector<int> > > OntoComps()`, которая на вход получает `Graph G` и на выход даёт вектор пар: связный граф `Graph H` и правило соответствий вершин H вершинам G `vector<int> l`.

2.3.4 Анализ случаев и покраска подграфов

Эта операция прописана в теле гравной функции `makeGr`. В цикле проверяются все подграфы из `vector<pair< Graph , vector<int> > > l` по схеме, указанной в приложении.

2.3.5 Поиск компонент двусвязности

За поиск компонент отвечает функция `GetBic()`. На вход она получает граф G, а на выход даёт список вершин одного из её крайних блоков.

Структурно, она преставляет собой последовательный запуск процедура `getBiColor` и функции `getBiComp`. Первая каждой вершине приписывает в отдельном списке порядок её посещения при поиске в глубину. Вторая рекурсивно по дереву сравнивает для кажой вершины значение порядка обхода из первой фукциии и минимум среди ёе соседей и передаёт этот минимум своему предку. Если мы придем с ситуации, что переданное значение оказалось не больше среди минимума соседей, то мы нашли точку сочленения, а значит все потомки этой вершины в остовном дереве лежат во одной компоненте двусвязности и образуют компоненту двусвязности.

Алгоритм также вынесен в приложение.

2.3.6 Поиск элементарных компонент и покраска

Функция `findCl()` ищет некоторый индуцированный цикл в исходном двусвязнос графе поиском в глубину со временем.

Функция `findClique()` ищет максимальную клику, содержащую данный подграф.

Data: Элемент типа `Graph H`, вектор изначальных вершин множества `vector<int > A`
Result: Вектор вершин в максимальной клике `vector<int > A`

`queue q`
`bool u` \\ отметки о рассмотрении вершин H
Добавить всю $N_H(A)$ в очередь.
Отметить рассмотрёнными вершины H
while *q не пуста* **do**
  Взять ранее не помеченную вершину v из q из очереди и убрать её.
  Отметить рассмотрённой.
  if *v смежна с кажой вершиной A* **then**
    Добавить v в A. Добавить всех ранее не отмеченных соседей v в очередь
  end
end

Algorithm 1: Схема работы `findClique()`

Функция `findThetaT`, `findThetaC` ищут θ-графы способом, описанным в теоретической части для случаем треугольного цикла и в случае выявления нечётного цикла, отличного от треугольного. Их основная особенность – на выходе эти функции передают пару `pair< vector<int>, pair<int, int> >` – пару из списка вершин и "начала"и "конца"θ-графа для дальнейшей покраски.

Покраска осуществляется согласно алгоритму при помощи функций `ColorTheta` Для покраски компонент связности с θ-графом с выделенными началом и концом, а так же с нечётными циклами с различными списками и `ColorCycle` для компонент с чётными циклами.

Итоговую сборку и объединение всех раскрасок на компонентах связности обеспечивает функция `CorrInj`, принимающая на вход граф G с раскрашенным подграфом H и правилом совмещения вершиу u и возвращающая вложенную раскраску G.

3 Приложение

3.1 Схема работы makeGr

```
Data: Элемент типа vector<pair< Graph , vector<int> > > A
Result: Вектор правильно раскрашенных vector<pair< Graph , vector<int> > > A
for i = 0 i < A.size() do
    u = GetBic(A[i].first) \\ Выделим компоненту двусвязности
    if FullCheck(H) then
        GreedyColor(H,0) \\ Покрасим жадным алгоритмом от вершины 0
        CorrInj(A[i].first, H, u) \\ Вставим раскраску H в G
    else
        if !RegularityCheck() then
            i=notRegVert(H) \\ Найдём нерегулярную вершину в H
            GreedyColor(H,v) \\ Покрасим жадным алгоритмом от вершины i
            CorrInj(A[i].first, H, u)
        else
            <vector> g = findCl() \\ Находим индуцированный цикл.
            if g.Size() == 0 then
                i=notRegVert(H)
                GreedyColor(H, i)
            else if g.Size() == 3 then
                pair<vector<int>, pair<int, int> > d
                d=FindThetaT(H,u) \\ Найдём в H θ-подграф
                ColorTheta(H, u, d.second.first, d.second.second) \\ Покрасим всю компоненту с
                  выделенным θ-графом
            end
            else if 2 | u.size() ∧ u.size() > 3 then
                ColorCycle(H,u) \\ Покрасим всю компоненту с выделенным чётным циклом
            end
            else if 2 ∤ u.size() ∧ u.size() > 3 then
                pair<vector<int>, pair<int, int> > d
                d=FindThetaC(H, u)
                Выделим из цикла θ-граф
                if (d.first.size() ≠ 0) ∧ (d.second.first ≠ d.second.second) then
                    ColorTheta(H, u, d.second.first, d.second.second) \\ Нашлась часть кроме цикла
                else
                    GreedyColor(H, 0, V) \\ Граф представляет собой цикл
                end
            end
            CorrInj(A[i].first, H, u) \\ Соединим всё вместе
        end
    end
end
```

Algorithm 2: Схема работы makeGr

3.2 Схема работы `getBiComp`

Data: Граф `Graph H`, вершина `int i`, числовая разметка графа в порядке обхода в глубину, массив посещений вершин `bool L`, беглый счётчик `int m`

Result:

Отметить в L i как посщённую.

`int M, N`

`vector <int> y`

for $j : (i, j) \in E(G)$ *и* j *не посещена* **do**

  `m=i` \\ Передать счётчику занчение рассматриваемой вершины

  `y.pushback(getBiComp(H,j,L,m))` \\ Добавить в y значение от потомка

end

$M-$ минимум среди всех значений y. Если y пусто, возвратим $\#V(H) + 1$.

Вычислим минимум N среди всех значений времени обхода среди всех соседей i, вершины с номером m, исключая номер k. Если N пусто вернём $\#V(H) + 1$.

$H[k] = \min M, N$

`return H[k]`

Algorithm 3: Схема работы `getBiComp`

4 Литература

Список литературы

[1] S. Skulrattanakulchai Δ-*List vertex coloring in linear time.*, Information Processing Letters. — 2006. — Vol. 98, Iss. 3. — Pg. 101–106

[2] H.N. Gabow, S. Skulrattanakulchai *Coloring algorithms on subcubic graphs,*, Internat. J. Found. Comput. Sci. 15 (1) (2004) 21–40.