Второй курс, осенний 2018/19
Конспект лекций по алгоритмам

Собрано 5 сентября 2018 г. в 17:50

Содержание

Лекция #1: Паросочетания

4 сентября

1.1. Определения

Def 1.1.1. *Паросочетание (matching) – множество попарно не смежных рёбер M.*

Def 1.1.2. *Вершинное покрытие (vertex cover) –*
множество вершин C, что у любого ребра хотя бы один конец лежит в C.

Def 1.1.3. *Независимое множество (indpendent set) –*
множество попарно несмежных вершин I.

Def 1.1.4. *Клика (clique) – множество попарно смежных вершин.*

Def 1.1.5. *Совершенное паросочетание – паросочетание, покрывающее все вершины графа.*
В двудольном графе совершенным является паросочетание, покрывающее все вершины меньшей доли.

Def 1.1.6. *Относительно любого паросочетания все вершины можно поделить на*
- *покрытые паросочетанием (принадлежащие паросочетанию),*
- *не покрытые паросочетанием (свободные).*

Обозначения: Matching (M), Vertex Cover (VC или C), Independent Set (IS или I).

1.2. Сведения

Пусть дан граф G, заданный матрицей смежности g_{ij}. Инвертацией G назовём граф G', заданный матрицей смежности $g'_{ij} = 1 - g_{ij}$. тогда независимое множество в G задаёт клику в G', а клика в G задаёт независимое множество в G'.

Следствие 1.2.1. Задачи поиска max клики и max IS сводятся друг к другу.

Lm 1.2.2. Дополнение любого VC – IS. Дополнение любого IS – VC.

Следствие 1.2.3. Все три задачи поиска min VC, max IS, max clique сводятся друг к другу.

Утверждение 1.2.4. Задачи поиска min VC, max IS, max clique NP-трудны.

Утверждение было доказано в прошлом семестре в разделе про сложность.

1.3. Поиск паросочетания в двудольном графе

Def 1.3.1. *Чередующийся путь – простой путь, в котором рёбра чередуются в смысле принадлежности паросочетанию.*

Def 1.3.2. *Дополняющий чередующийся путь (ДЧП) – чередующийся путь, первая и последняя вершина которого не покрыты паросочетанием.*

Lm 1.3.3. Паросочетание P максимально $\Leftrightarrow \nexists$ ДЧП (лемма о дополняющем пути).

Доказательство. Пусть $\exists$ ДЧП $\Rightarrow$ инвертируем все рёбра на нём, получим паросочетание размера $|P+1|$. Докажем теперь, что если $\exists$ паросочетание $M\colon |M| > |P|$, то $\exists$ ДЧП. Для этого рассмотрим $S = M \triangledown P$. Степень каждой вершины в S не более двух (одно ребро из M, одно из P) $\Rightarrow$ S является объединением циклов и путей. Каждому пути сопоставим число a_i – разность количеств рёбер из M и P. Тогда $|M| = |P| + \sum_i a_i \Rightarrow \exists a_i > 0 \Rightarrow$ один из путей – ДЧП. ■

Лемма доказана для произвольного графа, но с лёгкостью найти ДЧП мы сможем только для двудольного графа.

Lm 1.3.4. Пусть G – двудольный граф, а P паросочетание в нём. Построим орграф $G'(G, P)$, в котором из первой доли во вторую есть все рёбра G, а из второй в первую долю только рёбра из P. Тогда есть биекция между путями в G' и чередующимися путями в G.

Lm 1.3.5. Поиск ДЧП в $G \Leftrightarrow$ поиску пути в G' из свободной вершины в свободную.

Следствие 1.3.6. Мы получили алгоритм поиска максимального паросочетания M за $\mathcal{O}(|M| \cdot E)$:

0. $P \leftarrow \varnothing$
1. Попробуем найти путь dfs-ом в $G'(G, P)$
2. `if` не нашли $\Rightarrow$ M максимально
3. `else goto 1`

1.4. Реализация

Важной идеей является применение ДЧП к паросочетания на обратном ходу рекурсии.

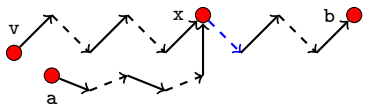
```
1 def dfs(v):
2   used[v] = 1 # массив пометок для вершин первой доли
3   for x in graph[v]: # рёбра из 1-й доли во вторую
4     if (pair[x] == -1) or (used[pair[x]] == 0 and dfs(pair[x])):
5       pair[x] = v # массив пар для вершин второй доли
6       return True
7   return False
```

Граф G' в явном виде мы нигде не строим. Вместо этого, когда идём из 1-й доли во вторую, перебираем все рёбра (`v` $\rightarrow$ `x in graph[v]`), а когда из 2-й в первую, идём только по ребру паросочетания (`x` $\rightarrow$ `pair[x]`).

1.5. Алгоритм Куна

Lm 1.5.1. В процессе поиска максимального паросочетания $\nexists$ ДЧП из $v \Rightarrow$ ДЧП из v уже никогда не появится.

Пусть появился ДЧП $p\colon v \rightsquigarrow u$ после применения ДЧП $q\colon a \rightsquigarrow b$. Тогда пойдём по p из v, найдём $x \in p$ – первую вершину в q. Из x пойдём по пути $a \leftrightarrow b$ по ребру паросочетания, дойдём до конца, получим пути $v \rightsquigarrow (a \vee b)$, который существовал до применения q.



■

Получили алгоритм Куна:

```
1 for v in range(n):
2   used = [0] * n
3   dfs(v)
```

1.6. Оптимизации Куна

Обозначим за k размер максимального паросочетания. Сейчас время работы алгоритма Куна $\mathcal{O}(VE)$ даже для $k = \mathcal{O}(1)$. Будем обнулять пометки `used[]` только, если мы нашли ДЧП.

```
used = [0] * n
for v in range(n):
  if dfs(v):
    used = [0] * n
```

Алгоритм остался корректным, так как, между успешными запусками `dfs` граф G' не меняется. И теперь работает за $\mathcal{O}(kE)$.

Следующая оптимизация – "жадная инициализация". До запуска Куна переберём все рёбра и те из них, что можем, добавим в паросочетание.

Lm 1.6.1. Жадная инициализация даст паросочетание размера $\geqslant \frac{k}{2}$.

Доказательство. Если мы взяли ребро, которое на самом деле не должно лежать в максимальном паросочетании M, мы заблокировали возможность взять $\leqslant 2$ рёбер из M. ■

При использовании жадной инициализации у нас появляется необходимость поддерживать массив `covered[v]`, хранящий для вершины первой доли, покрыта ли она паросочетанием.

Попробуем теперь сделать следующее:

```
used = [0] * n
for v in range(n):
  if not covered[v]:
    dfs(v)
```

Код работает за $\mathcal{O}(V + E)$. И, если паросочетание не максимально, найдёт хотя бы один ДЧП. А может найти больше чем один... в этом и заключается последняя оптимизация "вообще не обнулять пометки": пока данный код находит хотя бы один путь, запускать его.

Замечание 1.6.2. Докажите, что если мы используем последнюю оптимизацию, "жадная инициализация" является полностью бесполезной.

Напоминание: мы умеем обнулять пометки за $\mathcal{O}(1)$. Для этого помеченной считаем вершины `v`: "`used[v] == cc`", тогда операция "`cc++`" сделает все вершины не помеченными.

1.7. Поиск минимального вершинного покрытия

Lm 1.7.1. $\forall M, VC$ верно, что $|VC| \geqslant |M|$

Доказательство. Для каждого ребра $e \in M$, нужно взять в VC хотя бы один из концов e. ■

Пусть у нас уже построено максимальное паросочетание M. Запустим `dfs` на G' из всех свободных вершин первой доли. Обозначим первую долю A, вторую B. Посещённые `dfs`-ом вершины соответственно A^+ и B^+, а непосещённые A^- и B^-.

Теорема 1.7.2. $X = A^- \cup B^+$ – минимальное вершинное покрытие.

Доказательство. Если бы из $a \in A^+$ было бы ребро в $b \in B^-$, мы бы по нему прошли, и b лежала бы в $B^+ \Rightarrow$ в $A^+ \cup B^-$ нет рёбер $\Rightarrow X$ – вершинное покрытие. Оценим размер X: все вершины из $A^- \cup B^+$ – концы рёбер паросочетания M, т.к. `dfs` не нашёл дополняющего пути. Более того это концы обязательно разных рёбер паросочетания, т.к. если один конец ребра паросочетания лежит в B^+, то `dfs` пойдёт по нему, и второй конец окажется в A^+. Итого $|X| \leqslant |M|$. Из этого и 1.7.1 следует $|X| = |M|$ и $|X| = \max$. ■

Следствие 1.7.3. $A^+ \cup B^-$ – максимальное независимое множество.

Замечание 1.7.4. Мы умеем строить $\min VC$ и $\max IS$ за $\mathcal{O}(V+E)$ при наличии максимального паросочетания.

Следствие 1.7.5. $\max M = \min VC$ (теорема Кёнига).

1.8. Обзор решений

Мы изучили алгоритм Куна со всеми оптимизациями.
Асимптотически время его работы $\mathcal{O}(VE)$, на практике же он жутко шустрый.
На графах $V, E \leqslant 10^5$ решение укладывается в секунду.

$\exists$ также алгоритм Хопкрофта-Карпа за $\mathcal{O}(EV^{1/2})$. Его мы изучим в контексте потоков.

В регулярном двудольном графе можно найти совершенное паросочетание за время $\mathcal{O}(V \log V)$.
Статья Михаила Капралова и ко.

1.9. Решения для произвольного графа

1.9.1. Обзор

Наиболее стандартным считается решение через сжатие "соцветий" (видим нечётный цикл – сожмём его, найдём паросочетание в новом цикле, разожмём цикл, перестроим паросочетание). Этот подход используют реализации Габова за $\mathcal{O}(V^3)$ и Тарьяна за $\mathcal{O}(VE\alpha)$. Подробно эту тему мы будем изучать на 3-м курсе.

Оптимальный по времени – алгоритм Вазирани, $\mathcal{O}(EV^{1/2})$.

Кроме этого есть два подхода, которые мы обсудим подробнее.

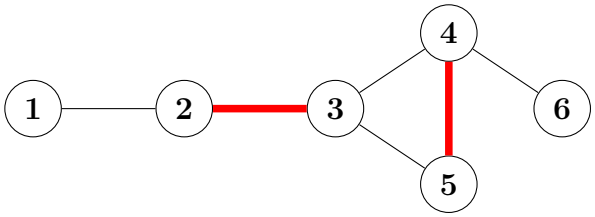
1.9.2. Матрица Татта

Рассмотрим матрицу Татта T. Для каждого ребра (i, j) элементы $t_{ij} = x_{ij}, t_{ji} = -x_{ij}$.
Остальные элементы равны нулю. Здесь x_{ij} – переменные.
Получается, для каждого ребра неорграфа мы ввели ровно одну переменную.
$\det T$ – многочлен от $n(n-1)/2$ переменных.
Теорема Татта: $\det T \not\equiv 0 \Leftrightarrow \exists$ совершенное паросочетание.
Чтобы проверить $\det T \equiv 0$, используем лемму Шварца-Зиппеля: в каждую переменную x_{ij} подставим случайное значение, посчитаем определитель матрицы над полем $\mathbb{F}_p$, где $p = 10^9 + 7$, получим вероятность ошибки не более n^2/p.
Время работы $\mathcal{O}(n^3)$, алгоритм умеет лишь проверять наличие совершенного паросочетания.
Алгоритм можно модифицировать сперва для определения размера максимального паросочетания, а затем для его нахождения.

Пример: $n = 3, E = \{(1,2),(2,3)\}, T = \begin{bmatrix} 0 & x_{12} & 0 \\ -x_{12} & 0 & x_{23} \\ 0 & -x_{23} & 0 \end{bmatrix}$, подставляем $x_{12} = 9, x_{23} = 7$, считаем $\det \begin{bmatrix} 0 & 9 & 0 \\ -9 & 0 & 7 \\ 0 & -7 & 0 \end{bmatrix} = 0 \Rightarrow$ с большой вероятностью нет совершенного паросочетания.

1.9.3. Читерский подход

Давайте на недвудольном графе запустим dfs из Куна для поиска дополняющей цепи...
При этом помечать, как посещённые, будем вершины обеих долей в одном массиве `used`.
С некоторой вероятностью алгоритм успешно найдёт дополняющий путь.



Если мы запускаем dfs из вершины 1, то у неё есть шанс найти дополняющий путь $1 \to 2 \to 3 \to 5 \to 4 \to 6$. Но если из вершины 3 dfs сперва попытается идти в 4, то он пометит 4 и 5, как посещённые, больше в них заходить не будет, путь не найдёт.
Т.е. на данном примере в зависимости от порядка соседей вершины 3 dfs или найдёт, или не найдёт путь. Если выбрать случайный порядок, то найдёт с вероятностью 1/2.
Это лучше, чем "при некотором порядке рёбер вообще не иметь возможности найти путь", поэтому первой строкой dfs добавляем `random_shuffle(c[v].begin(), c[v].end())`.

На большинстве графов алгоритм сможет найти максимальное паросочетание. $\exists$ графы, на которых вероятность нахождения дополняющей цепи экспоненциально от числа вершин мала.

Лекция #2: Паросочетания (продолжение)

11 сентября

2.1. Классификация рёбер двудольного графа

Дан двудольный граф $G = \langle V, E \rangle$. Задача – определить $\forall e \in E$, $\exists$ ли максимальное паросочетание $M_1 \colon e \in M_1$, а также $\exists$ ли максимальное паросочетание $M_2 \colon e \notin M_2$.
Иначе говоря, мы хотим разбить рёбра на три класса:

1. Должно лежать в максимальном паросочетании. `MUST`.
2. Может лежать в максимальном паросочетании, а может не лежать. `MAY`.
3. Не лежит ни в каком максимальном паросочетании. `NO`.

Решение: для начала найдём любое максимальное паросочетание M.
Если мы найдём класс `MAY`, то $\texttt{MUST} = M \setminus \texttt{MAY}$, $\texttt{NO} = \overline{M} \setminus \texttt{MAY}$.

Lm 2.1.1. $e \in \texttt{MAY} \Leftrightarrow \exists P$ – чередующийся относительно M путь чётной длины или чётный цикл, при этом $e \in P$.

Доказательство. Если ребро $e \in \texttt{MAY}$, то $\exists$ другое максимальное паросочетание $M' \colon e \in M \triangledown M'$.
Симметрическая разность, как мы уже знаем, состоит из чередующихся путей и циклов.

Посмотрим с другой стороны: если относительно M есть P – чередующийся путь чётной длины или чередующийся цикл, то $P \subseteq M$, так как и M, и $M \triangledown P$ являются максимальными, а каждое ребро P лежит ровно в одном из двух. ■

Осталось научиться находить циклы и пути алгоритмически. Для этого рассмотрим тот же граф G', на котором работает dfs из Куна. С чётными путями всё просто: все они начинаются в свободных вершинах, dfs из Куна, запущенный от всех свободных вершин *обеих долей* пройдёт ровно по всем рёбрам, которые можно покрыть чётными путями, и пометит их.

Lm 2.1.2. Ребро e лежит на чётном цикле $\Leftrightarrow$ концы e лежат в одной компоненте сильной связности графа G'.

2.2. Stable matching (marriage problem)

Сформулируем задачу на языке мальчиков/девочек. Есть n мальчиков, у каждого из них есть список девочек $bs[a]$, которые ему нравятся в порядке от наиболее приоритетных к менее. Есть m девочек, у каждой есть список мальчиков $as[b]$, которые ей нравятся в таком же порядке.
Мальчики и девочки хотят образовать пары.
Никто не готов образовывать пару с тем, кто вообще отсутствует в его списке.
И для мальчиков, и для девочек наименее приоритетный вариант – остаться вообще без пары.
Будем обозначать p_a – пара мальчика a или -1, q_b – пара девочки q или -1.

Def 2.2.1. *Паросочетание называется* **не** *стабильным, если* $\exists$ *мальчик* a *и девочка* b*: мальчику* a *нравится* b *больше чем* p_a *и девочке* b *нравится* a *больше чем* q_b.

Def 2.2.2. *Иначе паросочетание называется стабильным*

• **Алгоритм поиска: "мальчики предлагают, девочки отказываются"**

Изначально проинициализируем $p_a = bs[a].\texttt{first}$, далее, пока есть
два мальчика i, j: $b = p_i = p_j \neq -1$, Девочка b откажет тому из них, кто ей меньше нравится.
Пусть она отказала мальчику i, тогда делаем $bs[i].\texttt{remove\_first}(), p_i = bs[i].\texttt{first}$.

Теорема 2.2.3. Алгоритм всегда завершится и найдёт стабильное паросочетание

Доказательство. Длины списков $bs[i]$ убывают $\Rightarrow$ завершится. Пусть мальчик a и девочка b образуют не стабильность. Но это значит, что a перед тем, как образовать пару с p_a, предлагал себя b, а она ему отказала. Но зачем?! Ведь он ей нравился больше. Противоречие. ■

Аккуратная реализация даёт время $\mathcal{O}(\sum_i |bs[i]| + \sum_j |as[j]|)$: для каждой девочки b поддерживаем $s_b = \{a\colon p_a = b\}$, при присваивании $p_a \leftarrow b$ добавляем a в s_b и, если $|s_b| \geqslant 2$, вызываем рекурсивную процедуру $\texttt{откажиВсемКромеЛучшего}(b)$.

2.3. Венгерский алгоритм

Дан взвешенный двудольный граф, заданный матрицей весов $a\colon n \times n$, где a_{ij} – вес ребра из i-й вершины первой доли в j-ю вершину второй доли. Задача – найти совершенное паросочетание минимального веса.

Формально: найти $\pi \in S_n\colon \sum_i a_{i\pi_i} \to \min$.

Иногда задачу называют *задачей о назначениях*, тогда a_{ij} – стоимость выполнения i-м работником j-й работы, нужно каждому работнику сопоставить одну работу.

Lm 2.3.1. Если $a_{ij} \geqslant 0$ и $\exists$ совершенное паросочетание на нулях, оно оптимально.

Lm 2.3.2. Рассмотрим матрицу $a'_{ij} = a_{ij} + row_i + col_j$, где $row_i, col_j \in \mathbb{R}$.
Оптимальные паросочетания для a' и для a совпадают.

Доказательство. Достаточно заметить, что в $(f = \sum_i a_{i\pi_i})$ войдёт ровно по одному элементу каждой строки, каждого столбца $\Rightarrow$ если все элементы строки (столбца) увеличить на константу C, в не зависимости от π величина f увеличится на $C \Rightarrow$ оптимум перейдёт в оптимум. ■

Венгерский алгоритм, как Кун, перебирает вершины первой доли и от каждой пытается строить ДЧП, но использует при этом только нулевые рёбра. Если нет нулевого ребра, то $x = \min$ на $A^+ \times B^- > 0$. Давайте все столбцы из B^- уменьшим на x, а все строки из A^- увеличим на x.

	B^+	B^-
A^+		$-x$
A^-	$+x$	0

В итоге в подматрице $A^+ \times B^-$ на месте минимального элемента появится 0, в матрице $A^- \times B^+$ элементы увеличатся, остальные не изменятся. При этом все элементы матрицы остались неотрицательными. Рёбра из $A^- \times B^+$ могли перестать быть нулевыми, но они не лежат ни в текущем паросочетании, ни в дереве дополняющих цепей: $M \subseteq (A^- \times B^-) \cup (A^+ \times B^+)$, рёбра дополняющих цепей идут из A^+.

2.3.1. Реализация за $\mathcal{O}(V^3)$

Венгерский алгоритм = V поисков ДЧП.
Поиск ДЧП = инициализировать $A^+ = B^+ = \varnothing$ и не более V раз найти минимум $x = a_{ij}$ в $A^+ \times B^-$. Если $x > 0$, то пересчитать матрицу весов. Посетить столбец j и строку $pair_j$.

Чтобы быстро увеличивать столбец/строку на константу, поддерживаем row_i, col_j.
Реальное значение элемента матрицы: $a'_{ij} = a_{ij} + row_i + col_j$. Увеличение строки на $x\colon row_j \texttt{ += } x$.

Чтобы найти минимум x, а также строку i, столбец j, на которых минимум достигается, воспользуемся идеей из алгоритма Прима: $w_j = \min\limits_{i \in A^+} \langle a'_{ij}, i \rangle$. Тогда $\langle \langle x, i \rangle, j \rangle = \min\limits_{j \in B^-} \langle w_j, j \rangle$.

Научились находить (x, i, j) за $\mathcal{O}(n)$, осталось при изменении row_i, col_j пересчитать $w_j \colon j \in B^-$. col_j `+=` $y \Rightarrow w_k$ `+=` y. А row_i будет меняться только у $i \in A^- \Rightarrow$ на $\min_{i \in A^+}$ не повлияет.

Замечание 2.3.3. Можно выбирать min в множестве $w_j \colon j \in B^-$ не за линию, а используя кучи.

2.3.2. Псевдокод

Обозначим, как обычно, первую долю A, вторую B, посещённые вершины – A^+, B^+.
Также, как в Куне, если $x \in B$, то `pair[`x`]` $\in A$ – её пара в первой доли.
Строки – вершины первой доли (A). В нашем коде строки – i, v, x.
Столбцы – вершины второй доли (B). В нашем коде столбцы – j.
`pair[`$b \in B$`]` – её пара в A, `pair2[`$a \in A$`]` – её пара в B.

```
1. row ← 0, col ← 0
2. for v ∈ A
3.     A⁺ = {v}, B⁺ = ∅ // (остальное в A⁻, B⁻).
4.     w[j] = (a[v][j] + row[v] + col[j], v) // (минимум и номер строки)
5.     while (True) // (пока не нашли путь из v в свободную вершину B)
6.         ((z,i),j) = min{(w[j],j): j ∈ B⁻} // (минимум и позиция минимума в A⁺ × B⁻)
7.         // (i - номер строки, j - номер столбца ⇒ a[i,j] + row[i] + col[j] == z)
8.         for i ∈ A⁻: row[i] += z;
9.         for j ∈ B⁻: col[j] -= z, w[j].value -= z;
10.        // (в итоге мы уменьшили A⁺ × B⁻, увеличили A⁻ × B⁺, пересчитали w[j]).
11.        j перемещаем из B⁻ в B⁺; запоминаем prev[j] = i;
12.        if (x=pair[j]) == -1
13.            break // дополняющий путь: j,prev[j],pair2[prev[j]],prev[pair2[prev[j]]],...
14.        x перемещаем из A⁻ в A⁺;
15.        пересчитываем все w[j] = min(w[j], pair(a[x][j] + row[x] + col[j], x));
16.    применим дополняющий путь v ⇝ j, пересчитаем pair[], pair2[]
```

Текущая реализация даёт время $\mathcal{O}(V^3)$.
Внутри цикла `while` строки 6, 8, 9, 15 работают за $\mathcal{O}(V)$ каждая.
Из них 8 и 9 улучшить до $\mathcal{O}(1)$, храня специальные величины `addToAMinus`, `addToBMinus`.
Строки 6 и 15 можно улучшить до $\langle \mathcal{O}(\log V), deg[x] \cdot \mathcal{O}(1) \rangle$, применив кучу Фибоначчи.
Итого получится $\mathcal{O}(V(E + V \log V))$.

2.4. Покраска графов

2.4.1. Вершинные раскраски

Задача: покрасить вершины графа так, чтобы любые смежные вершины имели разные цвета.
В два цвета красит обычный dfs за $\mathcal{O}(V+E)$.
В три цвета красить NP-трудно. В прошлом семестре мы научились это делать за $\mathcal{O}(1.44^n)$.
Во сколько цветов можно покрасить вершины за полиномиальное время?

• **Жадность**

Удалим вершину v из графа $\to$ покрасим рекурсивно $G \setminus \{v\} \to$ докрасим v.
У вершины v всего deg_v уже покрашенных соседей $\Rightarrow$
в один из $deg_v + 1$ цветов мы её точно сможем покрасить.

Следствие 2.4.1. Вершины можно покрасить в $D{+}1$ цвет за $\mathcal{O}(V+E)$, где $D = \max_v deg_v$.

На дискретной математике будет доказана более сильная теорема:

Теорема 2.4.2. Брукс: все графы кроме нечётных циклов и клик можно покрасить в D цветов.

Кроме теоремы есть алгоритм покраски в D цветов за $\mathcal{O}(V+E)$.

На практике, если удалять вершину v: $deg_v = \min$ и докрашивать её в минимально возможный цвет, жадность будет давать приличные результаты. За счёт потребности выбирать вершину именно минимальной степени нам потребуется куча, время возрастёт до $\mathcal{O}((E+V)\log V)$.

Замечание 2.4.3. Иногда про покраску вершин удобно думать, как про разбиение множества вершин на независимые множества.

2.4.2. Рёберные раскраски

Задача: покрасить *рёбра* графа так, чтобы любые смежные рёбра имели разные цвета.
Попробуем для начала применить ту же жадность: удаляем ребро e из графа, рекурсивно красим рёбра в $G \setminus \{e\}$, докрасим e. У ребра e может быть $2(D{-}1)$ смежных, где $D = \max_v deg_v$. Значит, чтобы ребро e всегда получалось докрасить, в худшем, нашей жадности нужен $2D{-}1$ цвет, С другой стороны, поскольку рёбра, инцидентные одной вершине, должны иметь попарно разные цвета, Есть гораздо более сильный результат, который также подробнее будет изучен в курсе дискретной математики.

Теорема 2.4.4. Визинг: рёбра любого графа можно покрасить в $D{+}1$ цвет.

Доказательство теоремы представляет собой алгоритм покраски в $D{+}1$ цвет за $\mathcal{O}(VE)$.

При этом задача покраски в D цветов NP-трудна.

2.4.3. Рёберные раскраски двудольных графов

С двудольными графами всё проще. Сейчас научимся красить их в D цветов.
Покраска рёбер – разбиения множества рёбер на паросочетания. В последнем домашнем задании мы доказали, что в двудольном регулярном графе существует совершенное паросочетание.

Следствие 2.4.5. d-регулярный граф можно покрасить в d цветов.

Доказательство. Отщепим совершенное паросочетание, покрасим его в первый цвет. Оставшийся граф является $(d-1)$-регулярным, по индукции его можно покрасить в $d-1$ цвет. ■

Чтобы покрасить не регулярный граф, дополним его до регулярного.

• Дополнение до регулярного

1. Если в долях неравное число вершин, добавим новые вершины.
2. Пока граф не является D-регулярным (D – максимальная степень),
   в обеих долях есть вершины степени меньше D, соединим эти вершины ребром.
3. В результате мы получим D-регулярный граф, возможно, с кратными рёбрами.
   Кратные рёбра – это нормально, все изученные нами алгоритмы их не боятся.

Итого рёбра d-регулярного двудольного граф мы умеем красить за $\mathcal{O}(d \cdot \texttt{Matching}) = \mathcal{O}(E^2)$, а рёбра произвольного двудольного за $\mathcal{O}(D \cdot V \cdot VD) = \mathcal{O}(V^2D^2)$.

Поскольку в полученном регулярном графе есть совершенное паросочетание, мы доказали:

Следствие 2.4.6. Для $\forall$ двудольного G $\exists$ паросочетание, покрывающее все вершины G (в обеих долях) максимальной степени.

Раз такое паросочетание $\exists$, его можно попробовать найти, не дополняя граф до регулярного.

2.4.4. Покраска не регулярного графа за $\mathcal{O}(E^2)$

Обозначим A_D – вершины степени D первой доли, B_D – вершины степени D второй доли.
Уже знаем, что $\exists$ паросочетание M, покрывающее $A_D \cup B_D$.

1. Запустим Куна от вершин A_D, получили паросочетание P.
   Обозначим B_P покрытые паросочетанием P вершины второй доли.
2. Если $B_D \not\subseteq B_P$, чтобы покрыть $X = B_D \setminus B_P$ рассмотрим $M \triangledown P$. Каждой вершине из X в $M \triangledown P$ соответствует или ДЧП, или чётный путь из X в $Y = B_P \setminus B_D$.
3. Алгоритм: для всех $v \in X$ ищем путь или в свободную вершину первой доли, или в Y.

Чтобы оценить время работы, обозначим размер найденного паросочетания k_i и заметим, что нашли мы его за $\mathcal{O}(k_iE)$. Все k_i рёбер паросочетания будут покрашены и удалены из графа, то есть, $\sum_i k_i = E$. Получаем время работы алгоритма $\sum_i \mathcal{O}(k_iE) = \mathcal{O}(E^2)$.