Модуль 2. Ускорение перебора

Лекция 8

Параметризованные алгоритмы.
Часть 2

План лекции

1. Итерационное сжатие.
2. Решение задач на графах ограниченной древовидной ширины.
   - Идея алгоритма
   - Древовидная декомпозиция графа
   - Алгоритм динамического программирования

Итерационное сжатие

Метод основан на постепенном улучшении *имеющегося* решения. Для задач минимизации и задач максимизации схемы немного различаются.

Для задач минимизации в основе метода лежит использования *сжимающей процедуры*.

Определение

Сжимающая процедура — это алгоритм, получающий на входе пару (*x*,*s*), где *x* — входные данные, *s* — допустимое решение, и либо возвращающий допустимое решение s': |s'| < |s|, либо доказывающий, что s — минимальное решение.

Итерационное сжатие

Метод итерационного сжатия:

1. Построить начальное допустимое решение s.
2. Улучшать решение до тех пор, пока не получим оптимальное.

Очевидно, если сжимающая процедура является FPT-алгоритмом, то вся итерационная процедура также является FPT-алгоритмом.

Итерационное сжатие

Алгоритм итерационного сжатия для построения вершинного покрытия:

1. V' := ∅ // текущее множество покрытых вершин
2. U := ∅ // покрытие подграфа G[V'] — подграф, порождённый множеством V'
3. Последовательно перебрать вершины $v \in V \setminus V'$:
   - V' := V' ∪ {*v*}
   - U := U ∪ {*v*}
   - U := Сжатие(G[V'], U)
4. Вернуть U.

Итерационное сжатие

Осталось разработать процедуру сжатия.

На входе — подграф G' и его покрытие U.

Процедура *Сжатие* пытается модифицировать U, сохранив часть вершин ($Y \subset U$) и заменив оставшиеся S=U\Y на меньшее количество (|S|-1) вершин из V\U.

Как именно разделить U на Y и S?

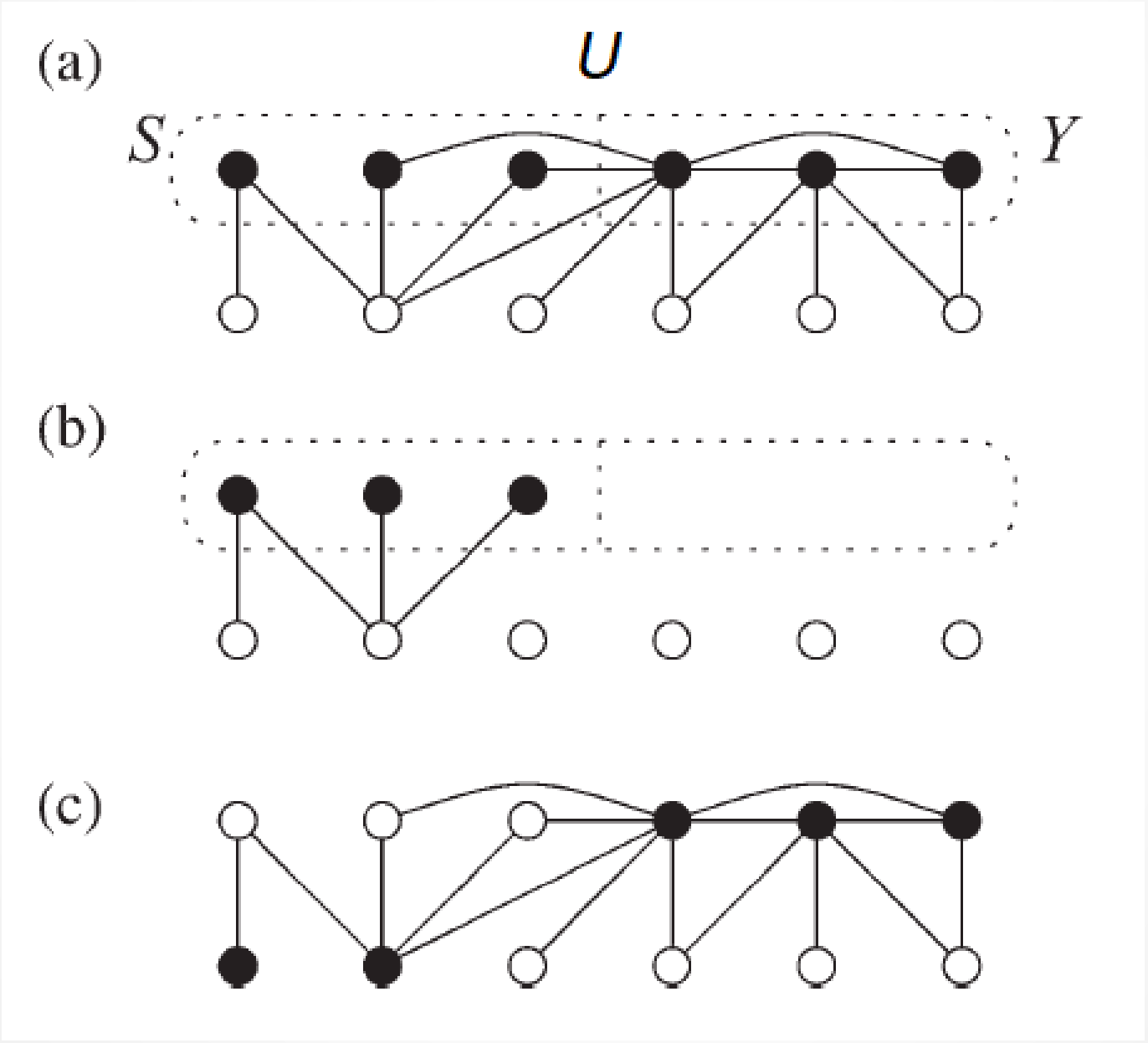
Переберём все $2^{|U|}(\leq 2^k)$ возможных варианта.

Итерационное сжатие

Рассмотрим фиксированное разделение $U = Y \cup S$.

- Вершины множества Y (вместе с инцидентными рёбрами) удаляем — они уже входят в покрытие.
- Поэтому рассматриваем G'[V'\Y]. S — вершиное покрытие на G'[V'\Y].
- Но все вершины множества S исключаем из рассмотрения, т. к. сейчас именно такой вариант рассматриваем.
- Поэтому, чтобы получить вершинное покрытие G'[V'\Y], необходимо для каждого ребра добавить в U ту из инцидентных вершин, которая не принадлежит S. Если оба конца какого-то ребра принадлежат S, то текущий вариант разделения $U = Y \cup S$ не позволяет улучшить покрытие.

Итерационное сжатие



Итерационное сжатие

Оценим сложность процедуры сжатия.

- Процедура вызывается $n=|V|$ раз.
- При каждом вызове перебираются $O(2^k)$ вариантов разделения U.
- Для каждого варианта надо проверить $O(m)$ рёбер.

Итоговая сложность: $O(2^k\, mn) \sim O(2^k\, n^3)$.

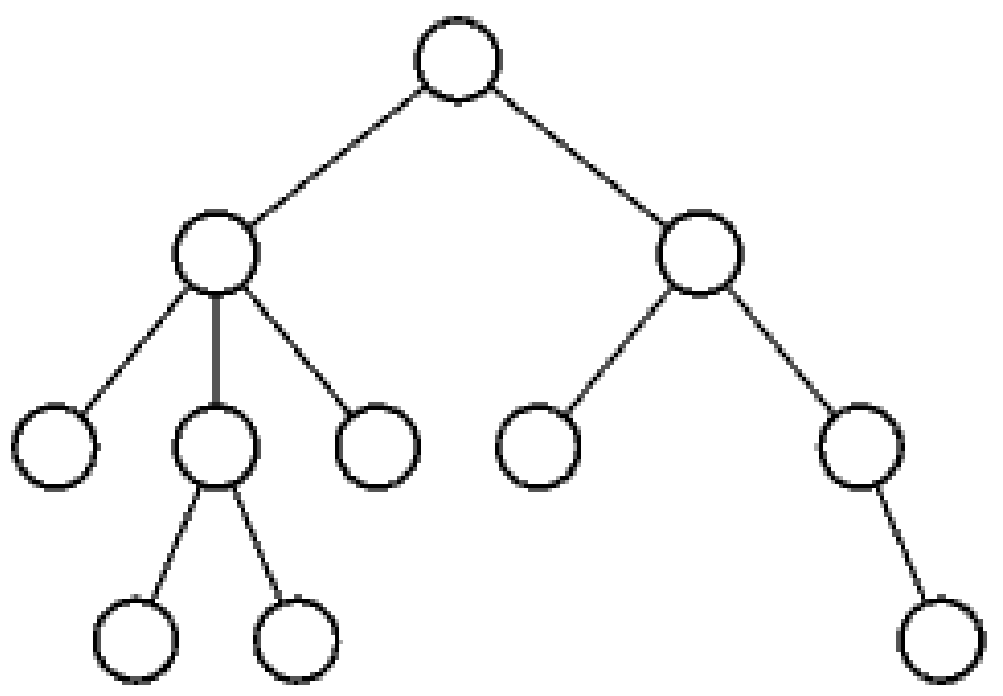
Древовидная декомпозиция

Метод древовидной декомпозиции является достаточно универсальным - подходит для широкого круга задач.

В качестве параметра используется характеристика, связанная не с искомым решением, а со свойствами входного графа.

Древовидная декомпозиция

Если *G* — *дерево*, то вершинное покрытие можно найти за линейное время O(*n*) с помощью параметрической редукции, последовательно применяя правило 2.



Древовидная декомпозиция

А если G — не дерево, но по структуре *близок* к дереву? Можем применить ту же идею.

Определение.

Древовидной декомпозицией графа $G(V,E)$ называется дерево $T(V_T, E_T)$, в котором каждая вершина X_i (*мешок*, *bag*) представляет подмножество вершин G и выполняются условия:

(1) покрытие вершин: каждая вершина G принадлежит хотя бы одному мешку;

(2) покрытие рёбер: для любого ребра e=(u,v) найдётся хотя бы один мешок, содержащий обе вершины {u,v};

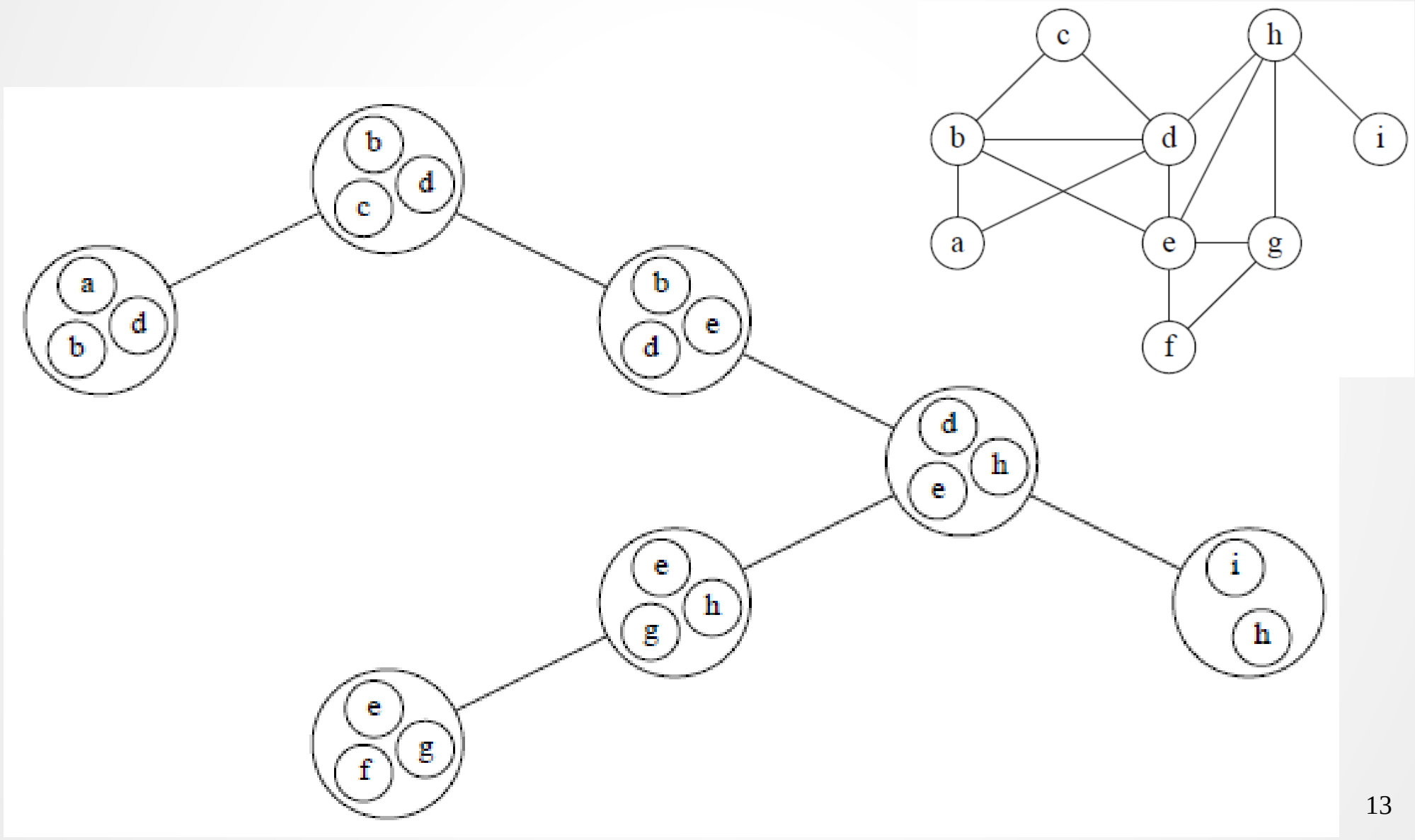
(3) условие согласованности: для любой тройки i,j,k: если на T вершина X_j лежит на пути между X_i и X_k, то $X_i \cap X_k \subseteq X_j$.

Условие (3) равносильно тому, что для любой $v \in V$ множество мешков, содержащих v, образует поддерево на T.

Ширина древовидной декомпозиции: $\max |X_i| - 1$.

Древовидная ширина графа tw(G) = минимальная ширина древовидной декомпозиции.

Древовидная декомпозиция



Древовидная декомпозиция

Есть проблема: построение оптимальной древовидной декомпозиции (и даже вычисление *tw*(*G*)) — NP-трудная задача.

Но для некоторых классов графов есть быстрые алгоритмы построения древовидной декомпозиции.

Теорема. *tw*(*G*)=1 <=> *G* — дерево.

Кроме того, для этой задачи найден FPT-алгоритм, эффективный при малых значениях tw(*G*).

Также: разработаны приближённые алгоритмы построения декомпозиций ширины ≤ 4 tw(G).

Древовидная декомпозиция

Но допустим, что для G древовидная декомпозиция T уже известна, и $\mathrm{tw}(G) \leq k$.

Как найти вершинное покрытие?

Существует алгоритм динамического программирования, строящий оптимальное вершинное покрытие за время $O(2^k k |V_T|)$.

Алгоритм вычисляет вершинные покрытия для подграфов, заданных объединениями мешков, начиная с листьев T и продвигаясь к внутренним узлам T.

Древовидная декомпозиция

Шаг 1. Создание таблиц.

Для каждого мешка X_i (узел дерева T) создать таблицу A_i.

$$A_i = \begin{array}{cccccc|c} x_{i_1} & x_{i_2} & \cdots & x_{i_{n_i-1}} & x_{i_{n_i}} & & m_i() \\ \hline 0 & 0 & \cdots & 0 & 0 & \Big\} & \\ 0 & 0 & \cdots & 0 & 1 & & \\ 0 & 0 & \cdots & 1 & 0 & & \\ 0 & 0 & \cdots & 1 & 1 & & 2^{n_i} \\ & & \vdots & & & & \\ 1 & 1 & \ldots & 1 & 0 & & \\ 1 & 1 & \ldots & 1 & 1 & \Big\} & \end{array}$$

Древовидная декомпозиция

Каждой строке j соответствует подмножество $C_i(j)$ элементов мешка, включаемых в покрытие.

$m_i(j)$ = минимальная мощность покрытия на G, содержащего все вершины из $C_i(j)$ и не содержащего ни одной вершины из $X_i \setminus C_i(j)$.

Если таких покрытий нет, то $m_i(j) = +\infty$.

Древовидная декомпозиция

Шаг 2. Инициализация таблиц.

Для каждого мешка X_i заведём таблицу и для каждой строки установим $m_i(j)$:

Если j-я строка задаёт допустимое подпокрытие $C_i(j)$, то $m_i(j) = |C_i(j)|$.

Иначе $m_i(j) = +\infty$.

Древовидная декомпозиция

Шаг 3. Пересчёт таблиц.

Пересчитываем таблицы для всех узлов T, начиная с листьев.

Пусть X_i — родительский узел для X_p и $Y = X_i \cap X_p$.

Для каждого j увеличиваем $m_i(j)$ на величину минимального покрытия G[Y], расширяющего $C_i(j)$.

$$m_i(j) := m_i(j) + \min \{ m_p(r): \text{для } C_{p|Y}(r) = C_{i|Y}(j) \} - |C_{i|Y}(j)|$$

Здесь $C_{i|Y}(j)$ — ограничение $C_i(j)$ на множество Y.

Древовидная декомпозиция

Если у X_i несколько дочерних узлов X_p , то последовательно выполняем такую процедуру для всех таких X_p.

Во вспомогательную таблицу P_i запоминаем индекс дочернего узла (p) и строки (r), на которых достигается минимум.

Поднимаемся от листьев до корня.

Древовидная декомпозиция

Шаг 4. Построение минимального вершинного покрытия.

Пусть X_r — корень дерева декомпозиции.

Найдём в A_r строку j с минимальным значением $m_r(j)$.

По „1“ из данной строки определяем, какие из вершин мешка X_r надо включить в покрытие (U).

По сохранённым для данного мешка индексам (шаг 3) определяем, к какому узлу T далее перейти и какую строку взять.

Так проходим от корня до листа.

Древовидная декомпозиция

Данный алгоритм динамического программирования строит оптимальное вершинное покрытие за время $O(2^k k |V_T|)$ для графа G с $tw(G) \leq k$.

Древовидная декомпозиция

Такой подход (динамическое программирование на основе древовидной декомпозиции графа) работает для широкого круга задач. А именно — для задач, в которых требуется проверить для графа *G* выполнение условия (предиката) *P*, представимого в виде конечной формулы в монадической логике второго порядка (MSOL).

То есть *P* можно описать с помощью формулы, содержащей предметные переменные (соответствуют вершинам и рёбрам графа, или *подмножествам вершин и рёбер*), предикатные переменные, логические операции и кванторы. И длина формулы не зависит от количества вершин/рёбер в графе.

Древовидная декомпозиция

Формулы MSOL, определяющие некоторые свойства графа

Свойство графа	Предикат	Формула MSOL, определяющая предикат
Совпадение рёбер e_i и e_j	$e_i = e_j$	$(\forall v_1)(\text{Incident}(v_1, e_i)) \Leftrightarrow (\text{Incident}(v_1, e_j))$
Смежность вершин v_i и v_j	$\text{Adjacent}(v_i, v_j)$	$\neg(v_i = v_j)\,\&\,(\exists e_1)\ (\text{Incident}(v_i, e_1)\ \&$ $\&\ \text{Incident}(v_j, e_1))$
V_1 — независимое множество вершин	$\text{IndependentSet}(V_1)$	$(\forall v_2)(\forall v_3)((v_2 \in V_1\,\&\,v_3 \in V_1) \Rightarrow$ $\neg\text{Adjacent}(v_2, v_3))$
V_1 — вершинное покрытие	$\text{VertexCover}(V_1)$	$(\forall e_2)(\exists v_3)(v_3 \in V_1\,\&\ \text{Incident}(v_3, e_2))$
V_1 — клика	$\text{Clique}(V_1)$	$(\forall v_2)(\forall v_3)((v_2 \in V_1\,\&\,v_3 \in V_1) \Rightarrow$ $\text{Adjacent}(v_2, v_3))$
V_1 — доминирующее множество	$\text{DominatingSet}(V_1)$	$(\forall v_2)(v_2 \in V_1 \vee (\exists v_3)(v_3 \in V_1\,\&$ $\&\ \text{Adjacent}(v_2, v_3)))$
Вершинная k-раскраска	$\text{VertexColorable}(V_1, \ldots, V_k)$	$(\forall v_0)(v_0 \in V_1 \vee \ldots \vee v_0 \in V_k)\,\&$ $\text{IndependentSet}(V_1)\,\&\ldots$ $\&\,\text{IndependentSet}(V_k)$
E_1 — связность	$\text{Connected}(E_1)$	$(\forall V_2)(\forall V_3)(\neg(\exists v_4)(v_4 \in V_2))\vee$ $(\neg(\exists v_5)(v_5 \in V_3))\vee$ $(\exists v_6)(\neg(v_6 \in V_2)\,\&\,\neg(v_6 \in V_3))\vee$ $(\exists e_7)(\exists v_8)(\exists v_9)(e_7 \in E_1\,\&\,v_8 \in V_2\,\&$ $v_9 \in V_3\,\&\ \text{Incident}(v_8, e_7)\,\&$ $\text{Incident}(v_9, e_7))$
E_1 — гамильтонов цикл	$\text{HamCycle}(E_1)$	$\text{Connected}(E_1)\,\&\,(\forall v_2)(\exists e_3)(\exists e_4)$ $(e_3 \in E_1\,\&\,e_4 \in E_1\,\&\,\neg(e_3 = e_4)\,\&$ $\text{Incident}(v_2, e_3)\,\&\ \text{Incident}(v_2, e_4)\,\&$ $(\forall e_5)((e_5 \in E_1\,\&\,\text{Incident}(v_2, e_5)) \Rightarrow$ $(e_5 = e_3 \vee e_5 = e_4)))$

Древовидная декомпозиция

Теорема Курселя (1990 г).

Пусть P — предикат, представимый формулой в MSOL.

Для графа $G(V,E)$ с $\mathrm{tw}(G) \le k$ существует функция $f:\mathbb{N}\times\mathbb{N}\to\mathbb{N}$ и алгоритм, проверяющий истинность $P(G)$ за время $O(n\cdot f(|P|,k))$, $n = |V|$.

Для рассмотренных примеров P задаётся конечной формулой, поэтому алгоритм работает время $O(n\cdot f(k))$.

Задание 5

Реализовать для задачи о вершинном покрытии один из рассмотренных параметризованных алгоритмов, на свой выбор.

Входные данные:

- Граф $G(V,E)$. Формат: аналогичен заданию 2.
- Параметр k.