Вопросы. Алгоритмы

NP-полнота.

Определение: В теории алгоритмов *классом NP* (от англ. non-deterministic polynomial) называют множество задач разрешимости, решение которых возможно проверить на машине Тьюринга за время, не превосходящее значения некоторого многочлена от размера входных данных, при наличии некоторых дополнительных сведений (так называемого сертификата решения).

Эквивалентно класс NP включает задачи, которые можно за полиномиальное время решить на недетерминированной машине Тьюринга.

Примеры:

1. Задача о клике: по данному графу узнать, есть ли в нём клики (полные подграфы) заданного размера. Сертификат — номера вершин, образующих клику.

2. Определение наличия в графе гамильтонова цикла. Сертификат — последовательность вершин, образующих гамильтонов цикл.

3. Неоптимизационный вариант задачи о коммивояжёре (существует ли маршрут не длиннее, чем заданное значение k) — расширенный и более приближенный к реальности вариант предыдущей задачи. Сертификат - такой маршрут.

Определение: *NP-полная задача* — в теории алгоритмов задача с ответом «да» или «нет» из класса NP, к которой можно свести любую другую задачу из этого класса за полиномиальное время.

Таким образом, NP-полные задачи образуют в некотором смысле подмножество «типовых» задач в классе NP: если для какой-то из них найден «полиномиально быстрый» алгоритм решения, то и любая другая задача из класса NP может быть решена так же «быстро».

Более формальное определение: Язык L_2 называется *NP-трудным*, если любой язык из класса NP сводится к нему. Язык называют *NP-полным*, если он NP-труден, и при этом сам лежит в классе NP.

Доказательство NP-полноты задачи.

1. Необходимо свести задачу к *задачи принятии решения* (отвечает ДА или НЕТ). Задача оптимизации не проще, чем задача принятия решения, поэтому если ЗПР сложна, то соответствующая задача оптимизации также сложна.

2. Необходимо привести задачу к другой, для которой известна ее NP-полнота. Пример — задача о выполнимости схем.

(YA, Задача из экзамена ВШЭ. PROG. 2020) Задача о рюкзаке, варианты, методы решения. Теорема об алгоритмах с гарантированной абсолютной точностью. Жадные алгоритмы.

Сформулируйте задачу о рюкзаке. Почему она относится к классу NP-полных задач?

Классическая постановка задачи: Пусть имеется n предметов, каждый из которых имеет два параметра — масса w_i и ценность v_i. Также имеется рюкзак определённой грузоподъёмности W. Задача заключается в том, чтобы собрать рюкзак с максимальной ценностью предметов внутри, соблюдая при этом ограничение рюкзака на суммарную массу.

Другими словами, необходимо

$$\sum_{i=1}^{n} v_i x_i \to \max;$$

$$\sum_{i=1}^{n} w_i x_i \leq W,$$

где $x_i \in \{0, 1\}$.

Задача о рюкзаке относится к классу NP-полных, и для нее пока не найден полиномиальной алгоритм, решающий ее за разумное время.

При дополнительном ограничении на веса предметов, задачу можно решить за псевдополиномиальное время методами динамического программирования. Если вес каждого предмета w_i является целым, то можно определить функцию $m[w]$, равную максимальной стоимости предметов общим весом w. Тогда

1. $m[0] = 0$;

2. $m[w] = \max(m[w - w_i] + v_i)$, где $w_i \leq w$.

На каждом шаге алгоритм перебирает n предметов, ответ на задачу соответствует значению $m[W]$, поэтому для время работы данного алгоритма составляет $O(nW)$. Поскольку W может зависеть экспоненциально от размера входных данных, то алгоритм является псевдополиномиальным.

Примечание: *Псевдополиномиальный алгоритм* — это полиномиальный алгоритм, проявляющий экспоненциальный характер только при очень больших значениях числовых параметров.

Определение: Алгоритм называется п*риближенным алгоритмом с гарантированной абсолютной точностью* K, если для любого примера I алгоритм находит значение $z^A(I)$ с отклонением от оптимума $z^*(I)$ не более K, то есть

$$z^*(I) - z^A(I) \leq K.$$

Обозначим через $T_A(n, W)$ трудоемкость алгоритма A для задачи с n предметами вместимостью рюкзака W.

Теорема. Пусть A — приближенный алгоритм с гарантированной абсолютной точностью K и трудоемкостью $T_A(n, W)$. Тогда алгоритм A для любого примера позволяет найти точное решение задачи о рюкзаке с той же трудоемкостью.

◀ Пусть пример I задается числами $p_1, ..., p_n, w_1, ..., w_n, C$.Построим новый пример I', положив $C' = C$, $w'_i = w_i$, $p'_i = (K+1)p_i$. Для обоих примеров оптимальные наборы x^*_i совпадают.

Для примера I' имеем: $z^*(I') - z^A(I') \leq K$, но $z^A(I') = (K+1)z^A(I)$ и $z^*(I') = (K+1)z^*(I)$, тогда $z^*(I) - z^A(I) \leq K/(K+1) \leq 1$. Так как $p_i \in \mathbb{Z}$, то $z^*(I) = z^A(I)$. ▶

Определение: Жадный алгоритм — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным.

Примеры: выбор заявок, покрытие точек единичными отрезками, поиск минимального остовного дерева, алгоритм Дейкстры.

Доказательство NP-полноты задачи о рюкзаке.

Можно переформулировать задачу о рюкзаке для того, чтобы она соответствовала требованиям к задачам из NP. Необходимо ответить ДА, если существует набор предметов суммарной стоимости $\geq p$ и суммарным весом $\leq W$.

Для доказательства необходимо доказать два факта:

1. Задача принадлежит классу NP.

2. Задача NP-трудна.

Доказательство принадлежности к NP.
В качестве сертификата возьмем удовлетворяющее условию задачи подмножество предметов P' с суммарным весом, не большим W и стоимостью не меньше p. Очевидно, оно удовлетворяет всем требованиям, налагаемым на сертификат. Проверяющая функция строится очевидным образом и работает за полиномиальное от размера входа время.

Доказательство принадлежности к NP-трудным.
Сведем задачу о сумме подмножества к задаче о рюкзаке. Пусть $f : (S, s) \to (P, W, p)$ — функция, осуществляющая сведение. Функция устроена так: для каждого числа $q \in S$ создадим предмет с массой и ценностью (q, q), а $W = p = S$. Если исходная задача о сумме подмножества имела решение S^*, то имеет решение и задача о рюкзаке. Если имеется быстрое решение задачи о рюкзаке, то быстро решается задача о сумме подмножества.

(SE) Поиск в глубину и в ширину. Топологическая сортировка ациклического ориентированного графа. Алгоритм Дейкстры для поиска кратчайших путей.

Имеется два способа представления графа $G = (V, E)$: как набора *списков смежных вершин* и как *матрицы смежности*. Оба способа применимы как для ориентированных, так и для неориентированных. Списки смежности обеспечивают компактное представление разреженных графов, то есть таких, для которых E заметно меньше, чем V^2. Представление в виде матрицы смежности предпочтительнее для плотных графов (E близко к V^2).

Поиск в ширину
Определение. Поиск в ширину (BFS, Breadth-first search) — алгоритм обхода графа, являющийся основой для многих важных алгоритмов для работы с графами. Классической задачей, которую решает поиск

в ширину, можно считать задачу нахождения длин кратчайших путей от исходной вершины s до остальных вершин.

Описание алгоритма

На каждом шаге алгоритм берет из начала очереди Q вершину v. Далее все непосещенные и смежные с v вершины помещаются в конец очереди, помечаются посещенными (например в массиве was), для них обновляется расстояние: $d[v.child] = d[v]+1$ и предок $p[v.child] = v$. Если очередь пуста, то алгоритм завершает работу.

Анализ времени работы

По построению алгоритма в очередь Q помещаются только непосещенные вершины, поэтому каждая вершина добавится в очередь не более одного раза. Добавление и удаление из очереди всех вершин потребует $O(V)$ времени. При этом для каждой вершины рассматривается не более $\deg(v)$ ребер, поэтому общее время на просмотр всех ребер не превосходит

$$T_e = \sum_{v \in V} \deg(v) = 2E = O(E).$$

Итоговое время работы составляет $O(V + E)$.

Доказательство корректности

Лемма. Если $Q = [v_1, v_2, ..., v_k]$, то $d[v_1] \leq d[v_2] \leq ... \leq d[v_k]$. Здесь $d[v]$ — расстояние от вершины s до v.

◀ Инвариант: Пусть v_i, $v_j \in Q$, тогда $|d[v_i] - d[v_j]| \leq 1$. После добавления вершины s в очередь Q утверждение очевидным образом выполняется. Пусть на i-м шаге из очереди удалена вершина v_i, а в очереди осталось a вершин с расстоянием d и b вершин с расстоянием $d + 1$. После добавления r смежных с v_i вершин в очереди оказалось a вершин с расстоянием d и $d + r$ вершин с расстоянием $d + 1$. Оба инварианта сохраняются ▶

Теорема. Алгоритм поиска в ширину в невзвешенном графе находит длины кратчайших путей до всех достижимых вершин.

◀ Пусть $\delta(v)$ — минимальное расстояние от вершины s до вершины v в G, $d[v]$ — расстояние, найденое алгоритмом.

Пусть BFS отработал на произвольном графе G и для каких-то вершин получил $d[v] > \delta(v)$. Пусть u — вершина с неправильным $d[u]$, но минимальным $\delta(u)$, $p[u] = v$ — ее предок, найденный алгоритмом, w — предок u на кратчайшем пути до s.

Заметим, что расстояние $d[w]$ найдено правильно в силу выбора u.

Значит $d[u] > d[w] + 1$.

При этом по построению алгоритма $d[u] = d[v] + 1$. Заменим в этом равенстве $d[u]$:

$$d[v] + 1 > d[w] + 1 \Rightarrow d[v] > d[w].$$

Значит вершина w была добавлена алгоритмом в очередь позднее и извлечена раньше, чем w. Но она смежна с u, значит в момент удаления w из очереди в случае $p[u] = null$ мы записали ее туда. При этом, если $p[u] \neq null$, то $p[u] \neq v$. Таким образом вершина v никак не может быть предком u. Противоречие ▶

Поиск в глубину

Определение. Поиск в глубину (Depth-first search, DFS) — один из методов обхода графа.

Описание алгоритма

Пусть задан граф $G = (V, E)$. Предположим, что в начальный момент времени все вершины окрашены в *белый* цвет. Выполним следующие действия:

1. Пройдем по всем вершинам $v \in V$ и выполним для них процедуру DFS(v), если v белая.

2. Описание процедуры DFS(u):

   - Перекрашиваем u в *серый* цвет.
   - Для всякой вершины w, смежной с u, запускаем процедуру DFS(w).
   - Перекрашиваем u в *черный* цвет.

Анализ времени работы

Алгоритм DFS будет вызван от каждой вершины не более одного раза (так как он вызывается только от белых вершин). Внутри процедуры DFS(u) рассматриваются ребра $\{e|\text{begin}(e) = u\}$. Всего таких ребер в графе $O(E)$, следовательно, время работы составляет $O(V + E)$.

Алгоритм поиска в глубину может применяться в следующих задачах:

- Поиск цикла в графе. Если граф ориентированный, то попытка пройти в серую вершину означает наличие цикла. Для неориентированного графа нужно только проверить, что рассматриваемое ребро не является тем, по которому мы пришли в вершину.

- Поиск компонент сильной связности.
- В топологической сортировке.
- Для поиска точек сочленения и мостов.

Топологическая сортировка

Определение. *Топологическая сортировка* ориентированного ациклического графа $G = (V, E)$ представляет собой такое линейное упорядочение всех его вершин, что если граф G содержит ребро (u, v), то u в таком упорядочении располагается раньше, чем v. Заметим, что при наличии цикла в графе такая сортировка невозможна.

Топологическая сортировка применяется для указания последовательности событий.

Алгоритмы топологической сортировки

Алгоритм Кана

Пусть $A(v)$ — множество всех вершин, из которых есть дуга в вершину v, а P — искомая последовательность вершин. Будем выполнять следующие действия:

Пока $|V| \neq 0$:

- Выбрать вершину v такую, что $A(v) = \varnothing$, $v \notin P$.
- Добавить v в P.
- Удалить вершину v из V вместе с исходящими дугами.

Алгоритм с использованием DFS

Лемма. Пусть $G = (V, E)$ — ациклический ориентированный граф, тогда $(u, v) \in E$ только если leave$[v]$ < leave$[u]$. Здесь leave$[v]$ — время выхода DFS из вершины v.

◀ Пусть мы только что открыли вершину u. Рассмотрим ребро (u, v). Вершина v не может быть серой, так как это означало бы наличие цикла. Если вершина v черная, то leave$[v]$ уже установлено, значит leave$[v]$ < leave$[u]$. Пусть v — белая, значит она потомок вершины u. Из этого следует, что вершина v будет закрыта раньше ▶

Теорема. Пусть $G = (V, E)$ — ориентированный ациклический граф. Тогда существует функция $\varphi : E \to \{1, ..., n\}$ такая, что если $(u, v) \in E$, то $\varphi(u) < \varphi(v)$.

◀ Пусть $\varphi(v) = n - \text{leave}[v] + 1$, тогда если в G есть ребро (u, v), то по доказанной лемме $\text{leave}[v] < \text{leave}[u]$, значит $\varphi(v) > \varphi(u)$. Таким образом, функция $\varphi(v) = n - \text{leave}[v] + 1$ удовлетворяет требованию теоремы ▶

Из определения функции $\varphi(v)$ следует алгоритм топологической сортировки:

1. Для всех белых вершин v запустим DFS(v). Выводим ответ.
2. Описание процедуры DFS(u):
   - Перекрашиваем u в серый цвет.
   - Для всякой белой вершины смежной с u запускаем DFS.
   - Красим u в черный и добавляем ее в начало ответа.

Алгоритм Дейкстры

Определение. Дан взвешенный ориентированный граф $G = (V, E)$ без дуг отрицательного веса. Найти кратчайшие пути от некоторой вершины a до всех остальных вершин.

Неформальное объяснение

Каждой вершине $v \in V$ сопоставим метку $d[v]$, равную минимальному известному расстоянию от a до v. На каждом шаге алгоритм будет пытаться уменьшить эту метку. Изначально $d[a] = 0$, $d[v \neq a] = \infty$.

Шаг алгоритма:

Если остались непосещенные вершины, то выбираем среди них вершину u такую, что $d[u]$ минимально среди всех непосещенных вершин. Для всех непосещенных соседей $v \in N(u)$ обновляем $d[v]$ если $d[v] > d[u] + w(u, v)$.

Обозначения:

$w(i, j)$ — вес ребра (i, j)
U — множество посещенных вершин
$d[u]$ — минимальное известное расстояние от a до u
$p[u]$ — кратчайший путь от a до u

Псевдокод

$d[v \neq a] \leftarrow \infty$, $d[a] \leftarrow 0$, $p[a] \leftarrow \varnothing$
Пока есть вершина не из U:
    Пусть $d[u] = min_{v \in V \setminus U} d[v]$

$u \to U$
Для всех соседей $v \in N(u) \setminus U$:
    Если $d[v] > d[u] + w(u, v)$, то
        $d[v] = d[u] + w(u, v)$
        $p[v] = p[u] + v$

Доказательство корректности

◀ Пусть $l(v)$ — длина кратчайшего пути от a до v. Докажем, что в момент посещения вершины z выполняется $d[z] = l[z]$.

База. При посещении a имеем $d[a] = 0 = l(a)$.

Переход. Пусть на очередном шаге алгоритм выбрал вершину $z \neq a$. Пусть P — кратчайший путь от a до z, а y — первая непосещенная вершина в P. Если вершина x предшествует y в пути P, то x посещена и по предположению индукции $d[x] = l[x]$. Заметим, что $l[y] = l[x]+w(x, y)$. Иначе был бы более короткий путь до y, а следовательно и до z.

Получается, что в момент посещения вершины x вершина y получила метку не больше, чем $d[x] + w(x, y)$, но из того, что $d[x] = l[x]$, получаем $d[y] \leq d[x] + w(x, y) = l[x] + w(x, y) = l[y]$.

Если теперь $z = y$, то переход доказан. Пусть $z \neq y$. Так как на данном шаге алгоритма была выбрана вершина z, то $l[z] \leq d[z] \leq d[y] = l[y] \leq l[z]$ ▶

Сложность алгоритма

Сложность алгоритма Дейкстры зависит от способа нахождения вершины v с минимальным $d[v]$, а также от способа хранения меток и способа хранения непосещенных вершин.

В простейшей реализации метки храним в массиве, значит поиск вершины на очередном шаге алгоритма займет $O(V)$ времени. При этом каждая вершина посещается один раз, и для нее рассматриваются ребра, ведущие к соседним вершинам. Таким образом на обработку всего графа потребуется $O(V^2 + E)$ времени. При этом оценка на ребра $E \leq V^2$ дает более простую асимптотику $O(V^2)$.