Лекция по алгоритмам #14
Тема: алгоритм Борувки, алгоритм Гольдберга

9 декабря

Собрано 28 декабря 2014 г. в 23:52

Содержание

1 Алгоритм Борувки построения MST

1.1 Обычный вариант

G - взвешенный связный неориентированный граф. Ищем остовное дерево минимального веса. Введем новую весовую функцию - для каждого ребра будем вместо веса брать пару $\langle w, e\rangle$, где w - вес ребра, e - индекс ребра. Теперь все веса ребер различны.

Для каждой вершины берем ребро минимального веса, исходящее из него. Получаем граф G'.

Утверждение. *G' - подграф некоторого MST.*

Доказательство.

1. G' не содержит циклов. Если есть цикл $v_1 \xrightarrow{e_1} v_2 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} v_n \xrightarrow{e_n} v_1$, такой что $\forall i \in [1, n]$ e_i - ребро минимального веса, исходящее из v_i, то можно составить систему строгих неравенств
$$\begin{cases} w(e_2) < w(e_1) \\ \cdots \\ w(e_n) < w(e_{n-1}) \\ w(e_1) < w(e_n) \end{cases}$$
которая очевидно несовместна.
2. Значит G' - лес. Каждое дерево в G' - часть некоторого MST по лемме о разрезе: добавляя новый узел u, мы всегда берем самое легкое ребро соединяющее $\{u\}$ и $V \setminus \{u\}$.

□

Добавляем ребра G' в ответ, после чего сжимаем все компоненты связности G' (ищем каждую компоненту поиком в глубину, перестраиваем граф) и повторяем алгоритм в новом графе. Делаем так, пока не останется одна вершина.

На каждой итрации тратим $\mathcal{O}(E)$. При этом, на каждой итерации $|V|$ уменьшается не менее чем в 2 раза $\Rightarrow 1 \leq$ #итераций $\leq \log V$. Общее время $\mathcal{O}(E \log V)$.

1.2 Улучшение

После сжатия можем сделать граф простым. Удаляем петли, среди кратных ребер оставлем одно с минимальным весом. Для этого сортируем ребра по номерам их концов цифровой сортировкой за $\mathcal{O}(E)$, затем одним проходом для каждой встречающйся пары вершин оставляем только ребро с минимальным весом.

Теперь время в худшем случае $T = V^2 + (\frac{V}{2})^2 + (\frac{V}{4})^2 + \cdots = \mathcal{O}(V^2)$.

Получаем оценку $\mathcal{O}(min(E \log V, V^2))$. Для этого же алгоритма можно доказать оценку $\mathcal{O}(E \log(\frac{V^2}{E} + 1))$.

2 Алгоритм Гольдберга

2.1 Вступление

Поиск кратчайших путей в графе с отрицательными ребрами.

Идея: строим потенциальную функцию $p(v)$, после чего вводим в графе новую весовую функцию $a \xrightarrow{e} b$: $w'_e = w_e + p(a) - p(b)$, кратчайшие пути остаются кратчайшими. Строим потенциалы так, чтобы все новые веса были положительны, и в графе можно было запустить алгоритм Дейкстры. Уже знаем алгоритм Джонсона: строим потенциалы с помощью алгоритма Форда-Беллмана, после чего запускаем алгоритм Дейкстры для каждой вершины. Время: $\mathcal{O}(FB + V \cdot Dijkstra) = \mathcal{O}(VE + VE \log V) = \mathcal{O}(VE \log V)$.

Алгоритм Гольдберга строит потенциальную функцию быстрее (за $\mathcal{O}(E\sqrt{V} \log N)$). Так как в оценке алгоритма Джонсона быстрее растет второе слагаемое, это не поможет быстрее решать задачу ASSP (поиск кратчайших путей между всем парами вершин). Зато сможем быстрее решать задачу SSSP (кратчайшие пути от одной вершины до всех) за $\mathcal{O}(Golberd + Dijkstra)$.

2.2 Простой вариант за $\mathcal{O}(VE)$

G - взвешенный ориентированный граф. $w \in [-1, +\infty) \cap Z$.

Изначально установим $p[v] = 0 \quad \forall v$ и будем менять эти значения.

Def Для некотрого множества вершин A и ребра (u, v), если $u \notin A, v \in A$, будем говорить, что это ребро входит в A; если $u \in A, v \notin A$, будем говорить, что это ребро исходит из A; если $u \in A, v \in A$, будем говорить, что это ребро лежит внутри A.

Def Назовем плохими вершинами те, из которых есть исходящие ребра веса -1.

Исправляем все плохие вершины поочередно: если v плохая, запускаем из нее DFS по обратным ребрам веса 0 или -1, получаем множество достигнутых вершин A. Если в A есть вершина u: $v \xrightarrow{-1} u$, то в графе есть отрицательный цикл и построить потенциальную функцию не удастся, заканчиваем. Иначе делаем $p[A]$ += 1 (здесь и далее это значит $p[v]$ += $1 \forall v \in A$). После этого w' для всех ребер исходящих из A увеличились на 1 (при этом вершина v перестала быть плохой), для всех ребер внутри A не изменились, для всех ребер входящих в A уменьшились на 1 (все эти ребра были положительны).

После применения операции ко всем вершинам, в графе не остается ребер отрицательного веса.

2.3 За $\mathcal{O}(E\sqrt{V})$

Слои

Рассмотрим подграф $G_p = \langle V, E' \in E \rangle$: $w(e) \in \{-1, 0\} \forall e \in E'$.

Выделим компоненты сильной связности. Если внутри компоненты есть ребра веса -1, значит в графе есть отрицательный цикл, заканчиваем. Иначе кратчайшие расстояния от любой точки до точек внутри компоненты равны, т. к. они лежат на цикле веса 0. Сжимаем компоненты связности, получаем ацикличный граф G^*_p.

Создадим фиктивную вершину s, добавим ребра веса 0 из нее во все вершины. Ищем расстояния от нее до всех вершин графа G_p^* (циклов нет, поэтому динамическим программированием за $\mathcal{O}(E)$). Множество вершин графа G разбивается на "слои" - множества вершин, расстояния до которых равны $0, -1, -2, \ldots$. Запоминаем слои и возвращаемся к исходному графу G.

Научимся исправлять все вершины одного слоя за $\mathcal{O}(E)$

Возьмем все плохие вершины одного слоя. Запускаем из них *DFS* (один) по обратным ребрам веса -1 или 0. Получаем множество A.

Утверждение. *Сделав $p[A]$ += 1 исправим все вершины данного слоя.*

Доказательство. Ребро веса -1 может идти только из вершины более раннего слоя (с меньшим по модулю расстоянием) в вершину более позднего. Поэтому в A попадут только вершины улучшаемого и предыдущих слоев. Значит все отрицательные ребра, исходящие из вершин улучшаемого слоя, будут исходящими из A. □

Научимся исправлять по одной вершине из каждого слоя за $\mathcal{O}(E)$

Можно легко найти путь p, в котором будет по одной вершине из каждого слоя. Достаточно взять любую вершину последнего слоя и пойти из нее по обратным ребрам веса -1.

Теперь последовательно берем вершины из p начиная с нулевого слоя, для каждой находим A с помощью *DFS* по обратным неположительным ребрам и делаем $p[A]$ += 1 (или заканчивам, если есть цикл отрицательного веса), но при этом сохраняем результат предыдущих поисков (наращивая A). Тогда по каждому ребру пройдем не более одного раза. Но нужно отслеживать ситуацию: после исправления предыдущей вершины (веса некоторых ребер уменьшились), из уже посещенных появились обратные ребра веса 0 и по ним тоже нужно пройти. Для этого будем использовать структуру данных, которая умеет хранить некоторое множество ребер (поддерживает операцию *Add*), уменьшать веса всех ребер на 1 (*DecreaseAll*) и выдавать множество ребер, которые в данный момент имеют вес 0 (*GetAllZeroes*).

Тогда во время *DFS* будем заносить туда все ребра положительного веса по которым пытались пройти. После операции $p[A]$ += 1, будем также делать *DecreaseAll* (веса ребер входящих в A действительно уменьшаются на 1, вместе с ними мы неправомерно уменьшаем веса ребер внутри A, но дальше они не будут нас интересовать, т.к. оба конца уже посещены). Вместе с очередным *DFS* будем также делать *GetAllZeroes* и пытаться пройти по полученным ребрам.

Описание структуры данных

Храним $vector\langle int\rangle q[M]$ - массив векторов, переменную *begin* - индекс ячейки где в данный момент хранятся ребра веса 0 (соответственно, в $begin + i$ хранятся ребра веса i).

```
Add(e): q[begin + e.w].push_back(e)
DecreaseAll: begin++
GetAllZeroes: return q[begin]
```

Все операции за $\mathcal{O}(1)$, требует $\mathcal{O}(E)$ памяти.

Итак, чтобы исправить по одной вершине из каждого слоя, повторяем последовательность команд:

```
DFS из v_i   v_i ∈ p
GetAllZeroes
Проход по полученным ребрам
Проверка отсутствия отрицательных циклов
p[A] += 1
DecreaseAll
```

Чтобы изменить потенциал для каждой вершины один раз, вместо $p[A_1] \mathrel{+}= 1, p[A_2] \mathrel{+}= 1, \cdots, p[A_M] \mathrel{+}= 1$ на соответствующей итерации делаем $p[A_1] \mathrel{+}= M, p[A_2 \setminus A_1] \mathrel{+}= M - 1, \cdots, p[A_M \setminus A_{M-1}] \mathrel{+}= 1$. Теперь в сумме мы тратим $\mathcal{O}(E)$. При этом веса ребер в конце будут нужными, а на промежуточных стадиях веса ребер внутри A могут быть неправильными, но они также не будут нами рассматриваться (оба конца уже посещены).

Пусть в графе k плохих вершин. Тогда в нем либо есть слой, который содержит не менее $\sqrt{k}$ плохих вершин, либо слоев больше, чем $\sqrt{k}$. В любом из случаев мы можем исправить не менее $\sqrt{k}$ плохих вершин за $\mathcal{O}(E)$.

Оценка

Утверждение. *Чтобы исправить k вершин, потребуется $\mathcal{O}(\sqrt{k})$ итераций.*

Доказательство. На каждом шаге уменьшаем число плохих вершин на $\geq \sqrt{k}$ (от текущего k).

Через $\leq \sqrt{\frac{k}{2}}$ шагов число плохих вершин уменьшится до $\frac{k}{2}$.

$$k - \sqrt{\frac{k}{2}} - \sqrt{\frac{k}{2}} - \cdots - \sqrt{\frac{k}{2}} \leq \frac{k}{2}$$

Число итераций до того, как k станет меньше 1:

$$\sum_{i:\ \frac{k}{2^i} \geq 1} \sqrt{\frac{k}{2^i}} \leq \sum_{i=1}^{\infty} \sqrt{\frac{k}{2^i}} = \sqrt{k} \sum_{i=1}^{\infty} \sqrt{\frac{1}{2^i}} = \mathcal{O}(\sqrt{k})$$

□

Итоговый вид алгоритма

```
while (True):
    G → G_p → G_p^* → слои
    k = количество плохих вершин в G
    if k == 0:
        break
    if ∃ большой слой:
        улучшаем все вершины этого слоя
    else:
        находим длинный путь
        улучшаем все его вершины
```

Суммарно используется $\mathcal{O}(E\sqrt{V})$ операций.

2.4 Общий случай

$G\langle V, E\rangle \quad w \in [-N, +\infty) \cap Z$.

Пусть $\delta = \lfloor \frac{N}{2} \rfloor$. Заметим, что за одно применение предыдущего алгоритма можем перейти к весам из $[-\delta, +\infty)$.

Будем считать плохими те вершины, из которых есть ребра веса $< -\delta$. После того, как получаем A (везде ходим по неположительным ребрам как и раньше), делаем $p[A] \mathrel{+}= \delta + 1$. При этом веса ребер входящих в A изменятся на $-\delta - 1$ (были ≥ 1, стали $\geq -\delta$), веса ребер иходящих из A изменятся на $\delta + 1$ (были $\geq -2\delta - 1$, стали $\geq -\delta$).

Соответственно, за $\mathcal{O}(\log N)$ итераций можем перевести веса в $[0, +\infty)$. Общее время $\mathcal{O}(E\sqrt{V}\log N)$.