Кратчайшие пути на графе, потоки в сетях

Кратчайшие пути на графе

Рассматриваемый алгоритм определяет расстояние между вершинами в простом орграфе с неотрицательными весами. К таким орграфам сводятся многие типы графов. Если граф не является простым, его можно сделать таковым, отбрасывая все петли и заменяя каждое множество параллельных рёбер кратчайшим ребром (т.е. ребром с минимальным весом) из этого множества, каждое неориентированное ребро заменяется парой ориентированных рёбер. Если граф не взвешен, то можно считать, что все рёбра имеют одинаковый произвольно выбранный положительный вес.

Пусть $\Gamma = (X, U, \Phi)$ - простой орграф, для каждого ребра $u \in U$ определён вес $w(u) \geq 0$. Найдём кратчайший путь между двумя произвольно выбранными вершинами. Несуществующие рёбра будем считать рёбрами с бесконечным весом. Сумму весов рёбер в пути будем называть весом или длиной пути. Обозначим $w_{ij} = w(u)$ - вес ребра $u = (x_i, x_j)$. Алгоритм поиска кратчайшего пути, начиная из вершины x_0 в вершину z, просматривает граф в ширину, помечая вершины x_j значениями-метками их расстояний от x_0. Метки могут быть временными и окончательными. Временная метка вершины x_j - это минимальное расстояние от x_0 до x_j, когда в определении пути на графе учитываются не все маршруты из x_0 в x_j. Окончательная метка x_j - это минимальное расстояние на графе от x_0 до x_j. Таким образом, в каждый момент работы алгоритма некоторые вершины будут иметь окончательные метки, а остальная их часть – временные. Алгоритм заканчивается, когда вершина z получает окончательную метку.

На первом шаге вершине x_0 присваивается окончательная метка 0, а для каждой из оставшихся $|X| - 1$ вершин присваивается временная метка ∞. На каждом шаге одной вершине с временной меткой присваивается окончательная и поиск продолжается дальше. На каждом шаге метки меняются следующим образом.

1. Каждой вершине x_j, не имеющей окончательной метки, присваивается новая временная метка – наименьшая из её временной и числа $w_{ij} + \text{окончательная метка } x_j$, где x_j – вершина, которой присвоена окончательная метка на предыдущем шаге.
2. Определяется наименьшая из всех временных меток, которая и становится окончательной меткой своей вершины. В случае равенства меток выбирается любая из них.

Пункты 1 и 2 мы повторяем до тех пор, пока вершина z не получит окончательной метки. Легко заметить, что окончательная метка каждой вершины – это кратчайшее расстояние от вершины x_0 до неё.

Рассмотрим работу этого алгоритма на примере. Найдём кратчайший путь из вершины x_0 в вершину x_6 (будем называть эту вершину z), для нижепредставленного графа

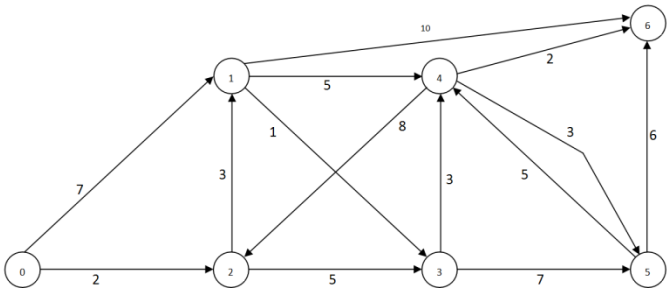

Рисунок 1

Процесс назначения меток вершинам графа на каждом шаге удобно представить в виде следующей таблицы.

Таблица 1

	x_0	x_1	x_2	x_3	x_4	x_5	x_6
0	**0**	∞	∞	∞	∞	∞	∞
1		7	**2**	∞	∞	∞	∞
2		**5**		7	∞	∞	∞
3				**6**	10	∞	15
4					**9**	13	15
5						12	**11**

Полужирным выделены окончательные метки, т.е. расстояния от x_0 до них. По такой таблице легко восстановить путь перемещения от z к x_0.

Реализация рассмотренной схемы поиска кратчайшего пути представлена в описанном ниже алгоритме. Граф $\Gamma = (X, U, \Phi)$ представляется матрицей весов $W_e = [w_{ij}]$, веса несуществующих рёбер полагаются равными ∞. Вектор $Mark[x]$ меток вершин устанавливает принадлежность вершины $x \in X$ постоянной (true) или временной (false) метке. Вектор $Dist[x]$ в алгоритме фиксирует текущие значения меток вершин. Вектор $Prev[x]$ позволяет восстановить в обратной последовательности вершины кратчайшего пути.

Листинг 1

```
for x ∈ X do
begin
  Mark[x]:=false;
  Dist[x]:=∞;
end;
y:=x0;
Mark[x0]:=true;
Dist[x0]:=0;
while not Mark[z] do
begin
  for x ∈ X do
    if not Mark[x] and Dist[x] > Dist[y] + w[y,z] then
    begin
      Dist[x]:=Dist[y]+w[y,x];
      Prev[x]:=y;
    end;
  Dist[y]:= min_{x∈X and Mark[x]=false} Dist[x];
  Mark[y]:=true;
end;
```

$Prev[x]$ указывает на вершину с окончательной меткой, ближайшую к вершине x. Последовательность вершин кратчайшего пути будет иметь следующий вид: $z, Prev[z], Prev\big[Prev[z]\big], \dots, x_0$, а значение $Dist[z]$ составит длину пути из x_0 в z. Очередная новая вершина, претендующая на постоянную метку обозначается через y.

Сложность алгоритма

Алгоритм обращается к телу цикла while не более $|X| - 1$ раз, и число операций, требующихся при каждом таком обращении, равно $O(|X|)$. Тогда сложность алгоритма составит $O(|X|^2)$. Интересно, что если требуется найти длины кратчайших путей от x_0 до всех вершин графа, то условие цикла while not $Mark[z]$ do надо заменить условием while $\vee_{x \in X}$ not $Mark[z]$ do. Сложность алгоритма при этом останется прежней.

Потоки в сетях

Определение. Транспортной сетью называется связный ориентированный граф без петель $\Gamma = (X, U, \Phi)$ с выделенной парой вершин x_0 и z. Вершина x_0 - начало транспортной сети, из которой дуги только выходят. Вершина z - конец транспортной сети, в которую дуги только входят. На множестве дуг $u \in U$ задана целочисленная функция $c(u) \geq 0$, где $c(u)$ - пропускная способность дуги.

Определение. Потоком по транспортной сети называется целочисленная функция $\varphi(u) \geq 0$, заданная на множестве дуг $u \in U$ и обладающая следующими свойствами:

$$\forall u \in U \; \varphi(u) \leq c(u)$$
$$\sum_{u \in U_x^+} \varphi(u) = \sum_{u \in U_x^-} \varphi(u)$$

где x - внутренняя вершина графа, т.е. $x \neq x_0$, $x \neq z$, U_x^+ - множество дуг, заходящих в вершину x, U_x^- - множество дуг, выходящих из вершины x. Второе свойство утверждает, что поток, входящий в вершину, равен выходящему потоку, т.е. поток не скапливается в вершинах. Введём следующие обозначения: $\varphi(z) = \sum_{u \in U_z^+} \varphi(u)$ и $\varphi(x_0) = \sum_{u \in U_{x_0}^+} \varphi(u)$, где $\varphi(z)$ – поток, входящий в вершину z, $\varphi(x_0)$ - поток, выходящий из вершины x_0.

Утверждение. $\varphi(z) = \varphi(x_0)$. Действительно, сумма $\sum_{x \in X}\big[\sum_{u \in U_x^+} \varphi(u) - \sum_{u \in U_x^-} \varphi(u)\big] = 0$, так как $\forall u \in U$ величина $\varphi(u)$ суммируется дважды: со знаком $+$ и со знаком $-$.

Определение. Пусть $A \subset X$ - множество вершин транспортной сети. $x_0 \notin A, z \in A$. Обозначим через U_A^+ множество дуг, входящих в A. Это множество будем называть разрезом транспортной сети. Величина $c(A) = \sum_{u \in U_A^+} c(u)$ называется мощностью разреза. Это максимально возможный поток, входящий в A по дугам разреза. $\varphi(A) = \sum_{u \in U_A^+} \varphi(u)$ - поток, входящий в A по дугам разреза. Ясно, что $\varphi(A) \leq c(A)$, так как $\forall u \in U \;\; \varphi(U) \leq c(u)$.

Утверждение. $\varphi(z) \leq c(a)$. Легко заметить, что не весь поток, входящий в A, скатывается в z. Часть потока может выходить из A. Значит, $\varphi(z) \leq \varphi(A)$, но $\varphi(A) \leq c(A)$, а значит $\varphi(z) \leq c(a)$.

Теорема Форда-Фалкерсона. Максимальный поток по транспортной сети равен мощности минимального разреза, т.е. $\max\limits_{\varphi} \varphi(z) = \min\limits_{\varphi} c(A)$.

Доказательство. Проведём конструктивное доказательство этой теоремы. Т.е. составим алгоритм определения максимального потока по сети. Алгоритм состоит из двух частей: насыщение и распределение потока.

1. Насыщение потока. Поток называется насыщенным, если любой путь из x_0 в z содержит дугу $u \in U$, для которой $\varphi(u) = c(u)$.
   1.1. Зададим произвольный начальный поток. Например, нулевой на всех дугах: $\forall u \in U$ $\varphi(u) = 0$
   1.2. Поиск пути из x_0 в z. Если путь найден, то переход к пункту 1.3. Если путь не найден, то переход к пункту 1.5.
   1.3. Увеличиваем поток по найденному пути таким образом, чтобы одна из дуг стала насыщенной.
   1.4. Условно разрываем насыщенную дугу и переходим к пункту 1.2.
   1.5. Сеть насыщена.
2. Перераспределение потока. Поток насыщен. Пометим рекурсивным образом все возможные вершины x_i сети.
   2.1. Вершину x_0 пометим 0.
   2.2. Пусть x_i - любая из уже помеченных вершин, y - произвольная непомеченная вершина, смежная с x_i. Вершину y помечаем $+i$, если данные вершины соединены ненасыщенным ребром, и помечаем $-i$, если соединены непустым ребром. Поле пометки вершин возможны два случая: вершина z оказалась либо помеченной, либо непомеченной.
   2.3. Вершина z оказалась помеченной. Значит, существует последовательность помеченных вершин от x_0 к z. В этой последовательности каждая последующая вершина помечена номером предыдущей. Определим на дугах новый поток, увеличивая на единицу поток на дугах, ориентированных по направлению движения от x_0 к z и уменьшая поток на единицу на дугах, направленных против этого движения. Заметим, что поток можно увеличивать (уменьшать) на прямых (обратных) дугах настолько, пока одна из дуг не станет насыщенной (пустой). Далее вновь переходим к пометке вершин (п. 2.1). Выполненное перераспределение потока сохраняет все его свойства и увеличивает на единицу поток в вершину z. Таким образом, пометка вершины z позволяет увеличить поток как минимум на единицу, а значит, алгоритм конечен, т.е. наступит момент, когда вершина z останется непомеченной.
   2.4. Вершина z осталась непомеченной. Пусть A^* - множество всех непомеченных вершин. Тогда дуги, входящие в эти вершины, насыщенные, а выходящие – пустые. Множество A^* определяет разрез, так как $x_0 \notin A^*, z \epsilon A^*$. Таким образом, мы нашли поток φ^* и разрез A^*, для которых выполняется $\varphi^*(A^*) = \mathrm{c}(A^*)$. Учитывая, что выходящие дуги из A^* пустые, то весь поток $\varphi^*(A^*)$ скатывается в z, т.е. $\varphi^*(z) = c(A^*)$.

Покажем, что φ^* - максимальный поток, а A^* определяет минимальный разрез. Так как для любого потока φ и любого разреза A справедливо $\varphi(z) \le c(A)$, то выполняется соотношение $\varphi^*(z) \le \max\limits_{\varphi} \varphi(z) \le \min\limits_{A} c(A) \le c(A^*)$.

Для найденного разреза и потока справедливо равенство $\varphi^*(z) \le c(A^*)$. Отсюда следует, что $\max\limits_{\varphi} \varphi(z) = \min\limits_{A} c(A)$. Рассмотренный алгоритм позволят найти именно максимальный поток φ^* и множество A^*, определяющее максимальный разрез.

Сложность алгоритма

Если следовать утверждению теоремы, то значение максимального потока можно найти прямым перебором всех разрезов. Подсчитаем общее число разрезов. Обозначим через $n = |X| - 2$, n - количество узлов в сети без вершин x_0 и z. Число сочетаний C_n^k равно числу разрезов, каждый из который содержит k вершин x_i и вершину z. Тогда число разрезов равно $C_n^0 + C_n^1 + \cdots + C_n^n = 2^n$, из которых надо выбрать один разрез. Следовательно, сложность полного перебора всех разрезов является экспоненциальной $O(2^{|X|})$.

Оценим количество операций в алгоритме Форда-Фалкерсона. Часть алгоритма, относящаяся к пометке вершин, требует перебора всех вершин, т.е. количества операций порядка $O(|X|)$. Каждое увеличение потока приводит к появлению дуги с нулевым или насыщенным потоком по ней. Такая дуга далее не участвует в пометке вершин. Всего дуг $|U|$, а значит, для окончания алгоритма требуется не более $O(|U| * |X|)$ операций приращения потока.