Взвешенные графы

Взвешенные графы

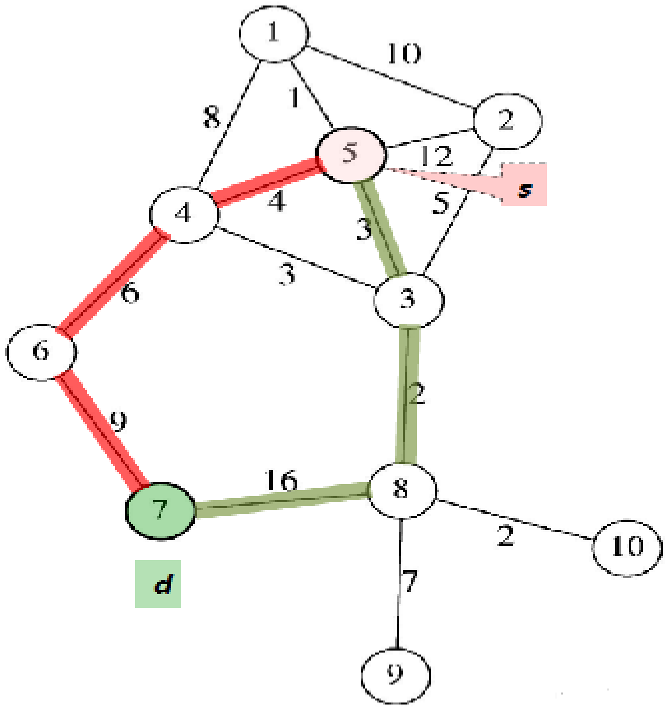
- Граф называется взвешенным, если его ребрам или вершинам приписаны некоторые значения, называемые весом или длиной, или стоимостью, или ценой. В каждом конкретном случае выбирается то слово, которое ближе подходит по смыслу задачи.
- Мы будем рассматривать случай, когда значения приписываются ребрам. Называть эти значения будем длинами ребер, хотя в реальности, эти значения могут быть и временем прохождения и ценой и чем-то другим, например, субъективной оценкой комфортности.
- Матрица смежности взвешенного графа содержит вместо единиц веса/длины ребер.

Длина пути

- Длина пути – это сумма длин ребер, входящих в этот путь.

- Длина пути (5, 4, 6, 7) = $w_{54} + w_{46} + w_{67} = 4 + 6 + 9 = 19$

- Длина пути (5, 3, 8, 7) = $3 + 2 + 16 = 21$



Задачи о кратчайшем пути

- **Задача о кратчайшем пути между парой вершин**
  Требуется найти кратчайший путь из заданной вершины *s* в заданную вершину *d*
- **Задача о кратчайших путях из заданной вершины во все**
  Найти кратчайшие пути из заданной вершины *s* во все
- **Задача о кратчайшем пути в заданный пункт назначения**
  Требуется найти кратчайшие пути в заданную вершину *v* из всех вершин графа
- **Задача о кратчайшем пути между всеми парами вершин**
  Требуется найти кратчайший путь из каждой вершины *u* в каждую вершину *v*

Алгоритм	Применение
Алгоритм Дейкстры	Находит кратчайший путь от одной из вершин графа до всех остальных. Алгоритм работает только для графов без ребер отрицательного веса ($w_{ij} \geq 0$)
Алгоритм Беллмана-Форда	Находит кратчайшие пути от одной вершины графа до всех остальных во взвешенном графе. Вес ребер может быть отрицательным
Алгоритм поиска A* (A star)	Находит путь с наименьшей стоимостью от одной вершины к другой, используя алгоритм поиска по первому наилучшему совпадению на графе
Алгоритм Флойда-Уоршелла	Находит кратчайшие пути между всеми вершинами взвешенного ориентированного графа
Алгоритм Джонсона	Находит кратчайшие пути между всеми парами вершин взвешенного ориентированного графа (должны отсутствовать циклы с отрицательным весом)
Алгоритм Ли (волновой алгоритм)	Находит путь между вершинами *s* и *t* графа, содержащий минимальное количество промежуточных вершин (трассировки электрических соединений на кристаллах микросхем и на печатных платах)
Алгоритмы Viterbi, Cherkassky, …	

Алгоритм Флойда — Уоршелла

- Алгоритм Флойда — Уоршелла — алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного графа без циклов с отрицательными весами с использованием метода динамического программирования.
- Этот алгоритм был одновременно опубликован в статьях Роберта Флойда (Robert Floyd) и Стивена Уоршелла (Stephen Warshall) в 1962 г., хотя в 1959 г. Бернард Рой (Bernard Roy) опубликовал практически такой же алгоритм, но это осталось незамеченным.

Динамическое программирование

- Динамическое программирование используется там, где оптимальное решение подзадачи меньшего размера может быть использовано для решения исходной задачи. В общем виде метод выглядит так:

  1. Разбиение задачи на подзадачи меньшего размера.
  2. Нахождение оптимального решения подзадач рекурсивно.
  3. Использование полученного решения подзадач для конструирования решения исходной задачи.

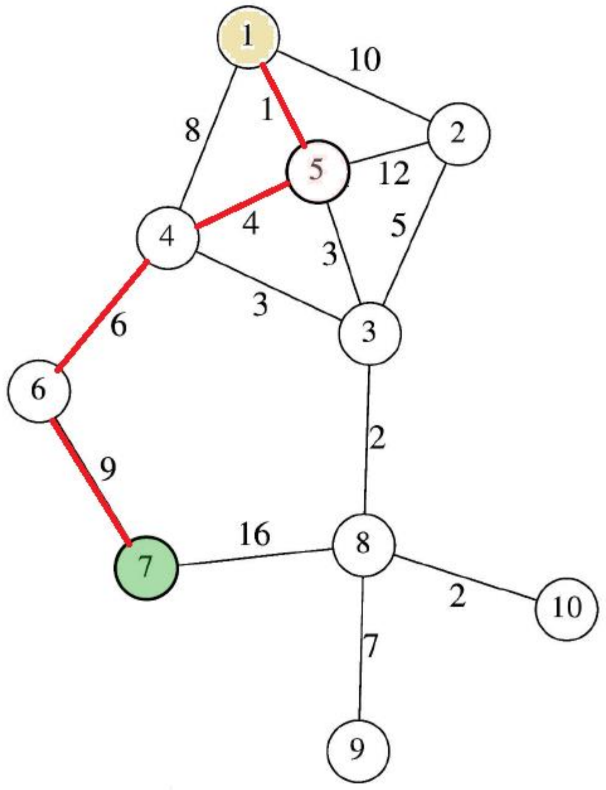
  Для нахождения кратчайших путей между всеми вершинами графа используется не перебор всех возможностей, что приведет к большому времени работы и потребует больше памяти, а восходящее динамическое программирование, то есть все подзадачи, которые впоследствии понадобятся для решения исходной задачи, просчитываются заранее и затем используются.

Первое свойство кратчайшего пути

- В основе алгоритма лежат два свойства кратчайшего пути графа. Первое: Имеется кратчайший путь $p_{1k}=(v_1,v_2,\ldots ,v_k)$ от вершины v_1 до вершины v_k, а также его подпуть $p'(v_i,v_{i+1},\ldots ,v_j)$, при этом действует $1 <= i <= j <= k$.
- Если p — кратчайший путь от v_1 до v_k, то p' также является кратчайшим путем от вершины v_i до v_j

Обоснование первого свойства кратчайшего пути

- Это можно легко доказать, так как стоимость пути p складывается из стоимости пути p' и стоимости остальных его частей. Так вот представив что есть более короткий путь p', мы уменьшим эту сумму, что приведет к противоречию с утверждением, что эта сумма и так уже была минимальной.



Путь с разрешенными промежуточными вершинами .

- Рассмотрим граф G с пронумерованными от 1 до n вершинами $\{v_1, v_2, \ldots, v_n\}$ и обозначим $\mathbf{d^k_{ij}}$ путь от v_i до v_j,, в котором промежуточные вершины могут быть только с номерами меньшими или равными k.

  Если k=0, то мы рассматриваем прямые соединения вершин друг с другом, так как множество разрешенных промежуточных вершин рано нулю. Если k=1 — мы рассматриваем пути, проходящие через вершину v_1, при k=2 — через вершины $\{v_1, v_2\}$, при k=3 — $\{v_1, v_3, v_3\}$ и так далее.

  Рассмотрим кратчайший путь p_{ij} с разрешенными промежуточными вершинами {1..k-1} стоимостью $\mathbf{d^{k-1}_{ij}}$. Теперь расширим множество на k- тый элемент, так что множество разрешенных вершин станет {1..k}. При таком расширении возможно 2 исхода:

Случай 1.

- Элемент k **не** входит в кратчайший путь p_{ij}, то есть от добавления дополнительной вершины мы ничего не выиграли и ничего не изменили, а значит стоимость кратчайшего пути d^k_{ij} не изменился, соответственно
- $\mathbf{d^k_{ij} = d^{k-1}_{ij}}$ — просто перенимаем значение до увеличения k.

Случай 2.

- Элемент k входит в кратчайший путь p_{ij}, то есть после добавления новой вершины в множество разрешенных, кратчайший путь изменился и проходит теперь через вершину v_k.
  Новый кратчайший путь разбит вершиной v_k на p_{ik} и p_{kj}, используем первое свойство, согласно ему, p_{ik} и p_{kj} также кратчайшие пути от v_i до v_k и от v_k до v_j соответственно. Значит

  $d^k_{ij} = d^k_{ik} + d^k_{kj}$

  А так как в этих путях k либо конечный, либо начальный узел, то они были вычислены до шага k:

- $\mathbf{d^k_{ij} = d^{k-1}_{ik} + d^{k-1}_{kj}}$

Второе свойство кратчайшего пути

- Значение d^k_{ij} в обоих случаях складывается из значений d для k-1, а значит имея начальные (k=0) значения для d, мы сможем рассчитать d для всех последующих значений k.
- При увеличении с k-1 до k, какое значение мы сохраним для d^k_{ik}? Минимумом значений случая 1 и 2, то есть смотрим дешевле ли старый путь или путь с добавлением дополнительной вершины.
- При k=n (n — количество вершин) мы получим оптимальные значения d для всех пар вершин.

рекуррентная формула для d^k_{ij} имеет вид:

d^0_{ij} — длина ребра $(i,\ j)$;

$$d^k_{ij} = \min(d^{k-1}_{ij},\ d^{k-1}_{ik} + d^{k-1}_{kj}).$$

Программа

На каждом шаге алгоритм генерирует матрицу W. Перед работой алгоритма матрица W заполняется длинами рёбер графа, или запредельно большими значениями, если ребер нет. Т.е. W на нулевом шаге - это матрица смежности, в которой вместо нулей стоят очень большие значения. При завершении работы алгоритма . матрица будет содержать длины кратчайших путей между всеми вершинами графа.

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      W[i][j] = min(W[i][j], W[i][k] + W[k][j])
```

Анализ времени работы алгоритма

- Алгоритм Флойда-Уоршелла используется для нахождения длины кратчайшего пути между всеми парами вершин во взвешенном графе за время $O(N^3)$.

Случай отрицательных циклов

- Если в графе есть циклы отрицательного веса, то формально алгоритм Флойда-Уоршелла к такому графу неприменим. Но на самом деле алгоритм корректно сработает для всех пар, пути между которыми никогда не проходят через цикл негативной стоимости, а для остальных мы получим какие-нибудь числа, возможно сильно отрицательные. Алгоритм можно научить выводить для таких пар некое значение, соответствующее -∞

  Кстати после отработки такого графа на диагонали матрицы кратчайших путей возникнут отрицательные числа — кратчайшее расстояние от вершины в этом цикле до неё самой будет меньше нуля, что соответствует проходу по этому циклу, так что *алгоритм можно использовать для определения наличия отрицательных циклов в графе*.

Алгоритм Форда-Беллмана. Расчет длины кратчайшего пути.

- Пусть дан неориентированный взвешенный граф G с ребрами неотрицательной длины и указана некоторая начальная вершина ***н*** и некоторая конечная вершина ***к***.

- Работая по алгоритму Форда-Беллмана, сначала ищутся длины кратчайших путей от вершины ***н*** до всех остальных вершин, потом строится сам кратчайший путь от выбранной вершины ***к***. Длины кратчайших путей будут вычисляться не сразу. Вначале работы алгоритма, мы присвоим вершинам некоторые числа, которые будем называть индексами, а по завершению работы алгоритма индексы вершин будут равны длинам кратчайших путей. Полагаем, что индекс вершины ***н*** равен 0, а всех остальных вершин ∞. На каждой шаге просматриваются все рёбра графа, и алгоритм пытается уменьшить значение индекса. Это возможно, если выполняется неравенство $L(c,y) < I(y) - I(c)$ (где $L(c,y)$ длина ребра (c,y), $I(y)$, $I(c)$- индексы вершин y, c, соответственно). Если это неравенство выполнено, то заменяем индекс вершины y на $L(c,y) + I(c)$. Расстановка индексов завершается, если данное неравенство не будет выполнено ни для одного ребра.

Построение кратчайшего пути после вычисления длины кратчайшего пути.

- Построение самого кратчайшего пути начинается от конечной вершины ***к.*** Сначала находим ребро, длина которого участвовала в вычислении индекса вершины ***к***, это ребро включаем в кратчайший путь, и переходим по этому ребру к вершине ***к'***, и ищем ребро, определившее индекс, уже вершины ***к'*** и т.д. Процедуру повторяем до тех пор, пока не достигнем вершины ***н.***
- Кратчайших путей может быть несколько, но по этому алгоритму будет построен только один из них.

Алгоритм Де́йкстры. Обозначения.

- Каждой вершине приписывается индекс – это длина пути от начальной вершины *а* до данной. Индекс вершины *v* будем обозначать *I(v)*.

- Вершины на определенном шаге, мы будем проходить. Если вершина пройдена, то ее индекс совпадает с длиной кратчайшего пути от нее до начальной вершины, если нет – то совпадение не обязательно.

- Одну из пройденных вершин мы выделим как текущую и обозначим буквой *с*.

Алгоритм Де́йкстры

- Шаг 1. Всем вершинам, за исключением первой, присваивается индекс равный бесконечности, а первой вершине – 0.
- Шаг 2. Первая вершина объявляется пройденной и текущей.
- Шаг 4. Рассматриваем все вершины, смежные текущей и не пройденные. Для этих вершин проверяем неравенство $L(c,y) < I(y) - I(c)$ (c- текущая вершина, y- рассматриваемая). Если это неравенство выполнено, то заменяем индекс вершины y на $I(y) = L(c,y) + I(c)$.
- Шаг 5. Среди не пройденных вершин ищется вершина с минимальным индексом. Ее мы проходим и объявляем текущей (предыдущая текущая вершина теряет свой статус «текущей вершины»).
- Шаг 6. Если текущая вершина совпадает с конечной, то завершаем работу.
- Шаг 7. Переходим к шагу 4.

Обоснование алгоритма Де́йкстры

- **Отметим, что алгоритм Дейкстры работает только на графах, у которых ребрам приписываются неотрицательные значения.**
- В силу связности графа наступит момент, когда мы дойдем до конечной вершины *b*. Вершины посещаются в некотором порядке. $v_1,\ldots, v_k$, где $v_1 = a$, $v_k = b$ Докажем, что в момент посещения вершины, ее индекс равен длине кратчайшего пути этой вершины до вершины *a*.
- Пусть *D*(*w*) длина кратчайшего пути от вершины *a* до вершины *w*. Доказательство проведем индукцией по номеру *n*, в последовательности $v_1,\ldots, v_k$.
- Базис индукции. Первой посещается вершина *a*. В этот момент $I(a)= D(a)=0$.

Обоснование алгоритма Де́йкстры 2

Индукционное утверждение. Пусть для всех вершин, посещенных ранее n, утверждение справедливо. Рассмотрим вершину v_n .

Предположим противное, т.е., что $I(v_n)\neq D(v_n)$,т.е. $I(v_n)>D(v_n)$, поскольку индекс не может быть меньше длины кратчайшего пути.

Пусть цепь $u_1,\ldots,u_m$, - кратчайшая цепь, соединяющая вершины a и v_n , т.е. $a= u_1$, $u_m = v_n$. Начальная часть этой кратчайшей цепи будет пройдена ранее вершины v_n, (хотя бы потому, что $a= u_1$).

Если вершина u_{m-1} пройдена ранее вершины v_n, то индекс вершины u_{m-1} равен кратчайшему пути до нее, т.е. $I(u_{m-1})=D(u_{m-1})$.

Но $D(v_n)= D(u_{m-1})+ L(u_{m-1}, v_n)$, отсюда получим, что на шаге 4, в тот момент, когда вершина u_{m-1} была текущей при проверке неравенства мы получим, что

$I(u_{m-1})+ L(u_{m-1}, v_n)< I(v_n)$, и заменим индекс $I(v_n)$ вершины v_n на

$I(u_{m-1})+ L(u_{m-1}, v_n)= D(u_{m-1})+ L(u_{m-1}, v_n) = D(v_n)$.

Получили противоречие.

Обоснование алгоритма Де́йкстры 3

Если вершина u_{m-1} не была пройдена ранее вершины v_n, то рассмотрим вершину u_i такую, что вершина u_{i-1} пройдена ранее вершины v_n, а вершина u_i - нет.

Тогда определенный на шаге 4 индекс вершины u_i , будет равен $I(u_i)=I(u_{i-1})+ L(u_{i-1}, u_i)= D(u_{i-1})+ L(u_{i-1}, u_i)$.

В этом случае, индекс $I(u_i)$ будет меньше $D(v_n)$, поскольку равен длине части кратчайшей цепи, соединяющей вершины a,v_n. Значит $I(u_i)< I(v_n)$.

Поскольку на шаге 5, мы выбираем для прохождения вершину с наименьшим индексом, то вершина v_n не может быть пройдена ранее вершины u_i. Опять получили противоречие.

Обоснование алгоритма Де́йкстры 4

Если вершина u_{m-1} не была пройдена ранее вершины v_n, то рассмотрим вершину u_i такую, что вершина u_{i-1} пройдена ранее вершины v_n, а вершина u_i - нет. Тогда определенный на шаге 4 индекс вершины u_i, будет равен $I(u_i)=I(u_{i-1})+ L(u_{i-1}, u_i)= D(u_{i-1})+ L(u_{i-1}, u_i)$. В этом случае, индекс $I(u_i)$ будет меньше $D(v_n)$, поскольку равен длине части кратчайшей цепи, соединяющей вершины a,v_n. Значит $I(u_i)< I(v_n)$. Поскольку на шаге 5, мы выбираем для прохождения вершину с наименьшим индексом, то вершина v_n не может быть пройдена ранее вершины u_i. Опять получили противоречие.

О применимости алгоритма Форда-Беллмана

- В программе рассматривается этот алгоритм на графах с неотрицательными весами.
- Если рассматривать ориентированный граф, то при вычислении разности *I(y)* - *I(c)* в качестве вершины y всегда берется конец дуги, в качестве вершины c начало дуги, поскольку движение по дуге возможно только в одном направлении. В этом случае веса могут быть и отрицательными, если граф не будет содержать циклов отрицательной длины, то кратчайший путь между вершинами может быть определен, и по данному алгоритму вычислен.
- Если рассматривать неориентированный граф с отрицательными весами, то алгоритм надо изменить так, что бы не зациклится на отрицательном ребре (y,c), беспрестанно меняя направление движения.