Алгоритм Флойда-Уоршелла

Это еще один алгоритм поиска кратчайшего маршрута во взвешенном графе. В отличие от алгоритма Дейкстры, алгоритм Флойда рассчитывает длины маршрутов между ЛЮБЫМИ двумя вершинами графа. Является одним из классических алгоритмов динамического программирования, когда решение основной задачи сводится к разбиению ее на элементарные подзадачи с последующим рекурсивным обращением к результатам.

Базовое утверждение:

Пусть дан маршрут P $\{v_1,v_2,\ldots,v_{n-1},v_n\}$ *и он является кратчайшим. В этом случае произвольный подмаршрут P`* $\{v_i,v_{i+1},\ldots,v_{m-1},v_m\}$ ***также является кратчайшим***.

Доказательство*: поскольку длина маршрута является суммой длин своих подмаршрутов, то в случае, если для любого из подмаршрутов можно найти меньшую длину, то длина исходного маршрута также уменьшится, что противоречит условию.*

Таким образом, если нам будет известно, через какую вершину проходит кратчайший путь между каждой парой вершин, мы можем рекурсивно из таких пар получить суммарный кратчайший маршрут между заданными вершинами.

Алгоритм Флойда состоит из двух фаз: в первой фазе создаются матрицы весов и достижимости. Во второй фазе происходит собственно поиск маршрутов.

Фаза 1.

Инициализация.

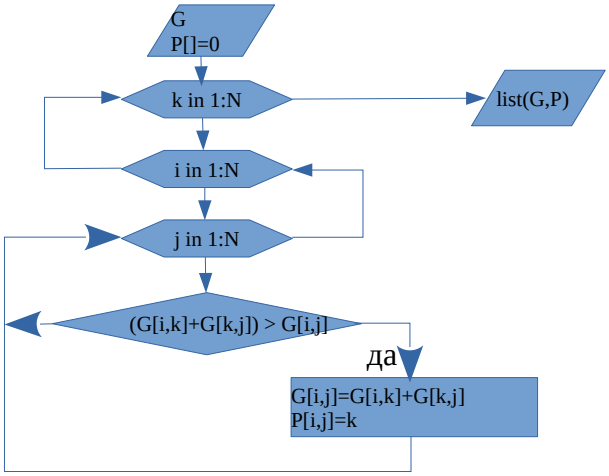
1. Модифицируется исходная матрица смежности: нулевые значение заменяются значением Inf, что означает бесконечно длинный путь (отсутствие пути)
2. Создается матрица достижимости P размерности N x N, где N — число вершин графа. Все элементы матрицы P равны нулю.

Основная часть.

Для всех значений **k** от 1 до N выполняется просмотр **всех пар** вершин графа G. Если длина пути $\mathbf{\{V_i,V_k\}+\{V_k,V_j\}}$, меньше длины пути $\mathbf{\{V_i,V_j\}}$, то длина пути $\mathbf{\{V_i,V_j\}}$ меняется на эту сумму (значение **G[i,j]** меняется на сумму **G[i,k]+G[k,j]**). Элемент матрицы достижимости **P[i,j]** в этом случае принимает значение **k** (что равносильно утверждению: «кратчайший маршрут между **i**-ой и **j**-ой вершиной проходит через вершину **k**»).

*Суть описанных действий: замена исходной матрицы смежности **G**, на «весовую матрицу» и «матрицу достижимости». Значением каждого элемента весовой матрицы является **min(G[i,j], G[i,k]+G[k,j])**, а значением P[i,j] соответственно **0** или **k**.*

Фрагмент алгоритма:



Алгоритм имеет константную кубическую сложность, однако если учесть, что рассчитываются сразу все возможные кратчайшие маршруты, то он имеет весьма широкие области применения, особенно в случае больших и «плотных» графов

Фаза 2.
Поиск маршрутов.
После получения матриц **G** и **P** мы можем искать кратчайшие маршруты. Матрица **G** содержит длины кратчайших путей между парами вершин графа, а соответствующий элемент матрицы **P** содержит номер некоторой промежуточной вершины в этом кратчайшем маршруте. Если между двумя смежными вершинами $\mathbf{V_i}$ и $\mathbf{V_j}$ нет более короткого пути, чем инцидентное ребро, то **P[i,j]==0**

Таким образом, используя эти две матрицы, мы получаем связный список, который можно использовать для рекурсивного поиска маршрута между любыми двумя вершинами $\mathbf{V_i}$ и $\mathbf{V_j}$.

Допустим, путь существует.
В этом случае элемент **G[i,j]** содержит отличное от ***Inf*** значение и маршрут либо состоит из вершин $\mathbf{V_i}$ и $\mathbf{V_j}$, если **P[i,j]==0** (т. е. $\mathbf{V_i}$ и $\mathbf{V_j}$ смежны и кратчайший путь есть инцидентное ребро), либо, (при **P[i,j]** отличном от нуля) из совокупности маршрутов **{** $\mathbf{V_i}$ **. . .** $\mathbf{V_k}$ **}** и **{** $\mathbf{V_k}$ **. . .**$\mathbf{V_j}$ **}**. Как вы понимаете, оба подмаршрута **{** $\mathbf{V_i}$ **. . .** $\mathbf{V_k}$ **}** и **{** $\mathbf{V_k}$ **. . .**$\mathbf{V_j}$ **}** ищутся рекурсивно. Условием выхода из рекурсии является смежность вершин $\mathbf{V_i}$ и $\mathbf{V_j}$ **(G[i,j]!=Inf & P[i,j]==0)**.

Некоторую техническую сложность составляет формирование вектора, описывающего маршрут, а именно исключение дубликатов вершин в том случае, если рекурсия опускается глубже, чем на 1 уровень.
Проще всего решить эту проблему написав рекурсивную часть, рассчитывающую только промежуточную часть маршрута, без вершин $\mathbf{V_i}$ и $\mathbf{V_j}$. Назовем эту функцию, например **R**. В этом случае весь маршрут будет состоять из:

- вершины $\mathbf{V_i}$
- **R(P,i,j)**
- вершины $\mathbf{V_j}$

Функция **R** будет возвращать пустой список в случае, если **P[i,j]==0** (*условие выхода из рекурсии*) и **c(R(P,i,k), k, R(P,k,j))** в ином случае.