**Алгоритм Флойда определения кратчайших путей между всеми парами вершин графа**

Алгоритм Флойда – это алгоритм поиска кратчайших путей между всеми вершинными парами графа.

Задан взвешенный орграф D=(V,E) с n вершинами и матрицей весов  Каждый элемент матрицы весов  равен весу дуги <,>, если такой дуги нет, то , а  Предположим, что заданный граф не содержит контуров отрицательной длины. Пронумеруем вершины графа . Обозначим Wk матрицу с элементами , каждый из которых равен длине кратчайшего пути из вершины i-ой в вершину j-ю, который может содержать в качестве промежуточных вершин только первые k вершин графа. Данное определение длины кратчайшего пути основывается на утверждении, что элемент матрицы смежности степеней К орграфа (k) Ak равен числу всех путей длины K из вершины  в вершину .

Если такого пути не существует, то , W0=W. По матрице W0 вычисляется матрица W1 и W2 и т.д., до тех пор, пока не будет определена матрица Wn, содержащая кратчайшие пути между всеми вершинами графа. Элементы матрицы Wk на k-ой итерации вычисляются следующим образом:

,

Где - длина кратчайшего пути из вершины  в вершину , в которой в качестве промежуточных используются первые К-1 вершины графа.

Для того, чтобы по окончании работы алгоритма можно было построить кратчайший путь, на каждой итерации вместе с матрицей Wk строится матрица Pk, каждый элемент которой  равен номеру вершины предшествующей вершине  в текущем ij пути.

На текущей операции выполняются операции:





Номера вершин, включаемых в кратчайший путь, определяются следующим образом:

(i,…., ),



и т.д.

Основные операции пошагового выполнения алгоритма.

1.Пронумеровать вершины графа целыми числами. Определить матрицу W0. Определить матрицу P0, и 

2. Если k=n, работа алгоритма закончена. Построенная матрица Wn- это матрица весов кратчайших путей между всеми парами вершин графа, определяемых с помощью матрицы Pn. Если k not equal n, то k=k+1, переход к шагу 3.

3. Вычисляем для всех i,j=1,…n элементы матрицы весов:



Если <, то .

Иначе .

4.Если для некоторого 1<q<n элемент с , то в графе имеется контур отрицательной длины и работа алгоритма завершается. Иначе перейти к шагу 2.

|  |
| --- |
|  |

**Алгоритм Дейкстры**

Алгоритм работает только для графов без рёбер отрицательного веса.

Неформальное объяснение

1. Присвоювання початкових значень. Виконати *l*(*v*) = 0 та вважати цю мітку постійною. Виконати *l*(*u*) = ∞ для всіх *u* ≠ *v* й уважати ці мітки тимчасовими. Виконати *x* = *v*, *M* = {*v*}.
2. Оновлення міток. Для кожної вершини *u* ∈ Г(*x*) \ *M* замінити мітки: *l*(*u*) = min{ *l*(*u*), *l*(*u*) + *w*(*x*, *u*) }, тобто оновлювати тимчасові мітки вершин, у які з вершини *x* іде дуга.
3. Перетворення мітки в постійну. Серед усіх вершин із тимчасовими мітками знайти вершину з мінімальною міткою, тобто знайти вершину *u*\* з умови *l*(*u*\*) = min{*l*(*u*)}, *u*∈*T*, де *T* = *V* \ *M*.
4. Уважати мітку вершини *u*\* постійною й виконати *M* = *M* ∪ {*u*\*}; *x* = *u*\* (вершину *u*\* включено в множину *М*).
5. а) Для пошуку шляху від *v* до *z*: якщо *x*=*z*, то *l*(*z*) – довжина найкоротшого шляху від *v* до *z*, зупинитись; якщо *x*≠*z*, то перейти до кроку 2.

б) Для пошуку шляхів від *v* до всіх вершин: якщо всі вершини отримали постійні мітки (включені в множину *М*), то ці мітки дорівнюють довжинам найкоротших шляхів, зупинитись; якщо деякі вершини мають тимчасові мітки, то перейти до кроку 2.

Алгоритм Дейкстри дає змогу обчислити довжину найкоротшого шляху від початкової вершини *v* до заданої вершини *z*. Для знаходження самого шляху потрібно лише збільшувати вектор вершин, з яких найкоротший шлях безпосередньо потрапляє в дану вершину. Для цього з кожною вершиною *u* графа *G*, окрім вершини *v*, зв’язують іще одну мітку - *θ*(*u*). Крок 2 модифікують так. Для кожної вершини *u*∈ Г(*x*) \ *M* якщо *l*(*u*) > *l*(*x*) + *w*(*x*,*u*), то *l*(*u*) = *l*(*u*) + *w*(*x*,*u*) та *θ*(*u*) = *x*, а ні, то не змінювати *l*(*u*) та *θ*(*u*). Коли мітка *l*(*u*) стане постійною, найкоротший 〈*v*, *u*〉-шлях буде потрапляти у вершину *u* безпосередньо з вершини *x*. Із постійних міток *l*(*u*) та *θ*(*u*) утворюємо вектори *l* і θ.

Алгоритм Белмана-Форда знаходить найкоротші шляхи від початкової вершини *v* до всіх решти вершини графа, але, на відміну від алгоритму Дейкстри, може використовуватись до графів із від’ємними вагами дуг. Обмеженням до використання алгоритму є наявність в графі циклів із від’ємною довжиною.

В процесі роботи алгоритму будується матриця *A*, елементи якої *aij* позначають довжину найкоротшого шляху від початкової вершини v у вершину *i*, що містить не більше *j* ребер. Шлях, що містить 0 ребер, існує тільки до вершини *v*. Таким чином, *ai0* дорівнює 0 при *i*=*v* та ∞ в іншому випадку.

Тепер розглянемо всі шляхи з *v* у *i*, що містять рівно *j* ребер. Кожен такий шлях є шлях з *j*-1 ребра, до якого додано останнє ребро. Якщо про шляхи довжини *j*-1 всі дані вже підраховано, то визначити *j*-й стовпець матриці не складає труднощів.

Алгоритм Белмана-Форда

1. Визначити *a*[*i*,0] = 0, якщо *i*=*v*, та ∞ для всіх інших вершин. Визначити *a*[*i*,*j*] = ∞ для всіх *j*∈[1; *n*-1], де *n* – кількість вершину графу.
2. Встановити *j*=1.
3. Для всіх вершин *u* виконати перевірку: якщо існує ребро (*u*,*x*) та *a*[*u*,*j*-1] + *w*(*u*,*x*) < *a*[*x*,*j*], то встановити *a*[*x*,*j*] = *a*[*x*,*j-1*] + *w*(*u*,*x*).
4. Збільшити *j* на 1. Якщо *j* дорівнює кількості вершин графу, то кінець алгоритму. Інакше перейти на пункт 2.

Для визначення самих найкоротших шляхів введемо матрицю *P*= ||*pij*||. Якщо елемент *aij* містить довжину найкоротшого шляху з *v* до *і*, який містить *j* дуг, то *pij* містить попередню вершину до *і* в одному з таких найкоротших шляхів. Отже, до пункту 3 алгоритму необхідно додати запис в матрицю *P*: *p*[*x*,*j*] = *u*, якщо *a*[*u*,*j*-1] + *w*(*u*,*x*) < *a*[*x*,*j*].

**Алгоритм Прима** — алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа.

Описание : Построение начинается с дерева, включающего в себя одну (произвольную) вершину. В течение работы алгоритма дерево разрастается, пока не охватит все вершины исходного графа. На каждом шаге алгоритма к текущему дереву присоединяется самое лёгкое из рёбер, соединяющих вершину из построенного дерева, и вершину не из дерева.

Способ представления графа и приоритетной очереди Асимптотика

Массив d, списки смежности (матрица смежности) O(V2)

Бинарная пирамида, списки смежности O((V + E)logV) = O(ElogV)

Фибоначчиева пирамида, списки смежности O(E + VlogV)

**Aлгоритм Крускала** — алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа.

Вначале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса.

Оценка : До начала работы алгоритма необходимо отсортировать рёбра по весу, это требует O(E × log(E)) времени. После чего компоненты связности удобно хранить в виде системы непересекающихся множеств. Все операции в таком случае займут O(E × α(E, V)), где α — функция, обратная к функции Аккермана. Поскольку для любых практических задач α(E, V) < 5, то можно принять её за константу, таким образом общее время работы алгоритма Крускала можно принять за O(E \* log(E)).

**Алгоритм фронта волны** поиска кратчайшего пути в ориентированном графе

Пусть задан ориентированный граф G(V, X).

Путь из vi в vj называется кратчайшим, если он содержит наименьшее число дуг по сравнению со всеми другими путями из vi в vj.

Будем искать кратчайший путь из вершины vi в vj.

1). Пометим вершину vi индексом "0". Она принадлежит фронту волны нулевого уровня W0(vi).

2). Рассмотрим образы вершины vi: Гvi. Пометим их индексом "1". Они принадлежат фронту волны первого уровня: W1(vi).

k). Рассмотрим образы всех вершин k-1ого уровня фронта волны: Г(Wk-1(vi)). Пометим непомеченные ранее вершины индексом k. Получим Wk(vi). Если через n-1 шагов мы не получим ни одном из множеств W1(vi),..., Wn-1(vi) вершины vj, то пути из vi в vj не существует.

Если на некотором шаге мы получили vjОWk(vi), то процесс окончен.

Пусть кротчайший путь содержит следующие вершины: vi " v(1) " v(2) " ..." v(k-1) " vj. Найдём номера вершин v(1),..., v(k-1):

v(k-1) О Wk-1(vi)ЗГ-1vj;

v(k-2) О Wk-2(vi)ЗГ-1v(k-1);

...

v(1) О W1(vi)ЗГ-1v(2);

Путей может быть несколько.

**Поиск в глубину** (англ. Depth-first search, DFS) — один из методов обхода графа. Алгоритм поиска описывается следующим образом: для каждой непройденной вершины необходимо найти все не пройденные смежные вершины и повторить поиск для них. Используется в качестве подпрограммы в алгоритмах поиска одно- и двусвязных компонент, топологической сортировки. Алгоритм поиска в глубину.

Пусть задан граф G = (V,E), где V — множество вершин графа, E — множество ребер графа. Предположим, что в начальный момент времени все вершины графа окрашены в белый цвет. Выполним следующие действия:

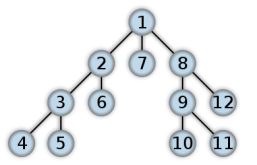
Из множества всех белых вершин выберем любую вершину, обозначим её v1.

Выполняем для неё процедуру DFS(v1).

Перекрашиваем её в чёрный цвет.

Повторяем шаги 1-3 до тех пор, пока множество белых вершин не пусто.

Процедура DFS (параметр — вершина )

Перекрашиваем вершину u в серый цвет.

Для всякой вершины w, смежной с вершиной u, выполняем следующие два шага:

Если вершина w окрашена в белый цвет, выполняем процедуру DFS(w).

Окрашиваем w в чёрный цвет.

**Поиск в ширину** (BFS, Breadth-first search) — метод обхода и разметки вершин графа.

Поиск в ширину выполняется в следующем порядке: началу обхода s приписывается метка 0, смежным с ней вершинам — метка 1. Затем поочередно рассматривается окружение всех вершин с метками 1, и каждой из входящих в эти окружения вершин приписываем метку 2 и т. д.

Если исходный граф связный, то поиск в ширину пометит все его вершины. Дуги вида (i, i+1) порождают остовный бесконтурный орграф, содержащий в качестве своей части остовное ордерево, называемое поисковым деревом.

Легко увидеть, что с помощью поиска в ширину можно также занумеровать вершины, нумеруя вначале вершины с меткой 1, затем с меткой 2 и т. д.

Алгоритм Террі знаходження маршруту

У зв’язаному графі зажди можна знайти такий маршрут, що зв’язує дві задані вершини *v* та *u*, якщо, виходячи з вершини *v* і здійснюючи послідовний перехід від кожної досягнутої вершини до суміжної з нею, керуватися такими правилами:

1. йдучи по довільному ребру, кожний раз відмічати напрямок, в якому воно було пройдене;
2. виходячи з деякої вершини *vi*, завжди рухатися тільки по тому ребру, яке не було пройдене або було пройдене у зворотному напрямку;
3. для кожної вершини *vi*, відмінної від *v*, відмічати те ребро, яке першим заходить у *vi*, якщо вершина *vi*, зустрічається вперше;
4. виходячи з деякої вершини *vi*, відмінної від *v*, по першому ребру, яке заходить у *vi*, рухатися лише тоді, коли немає інших можливостей.

Алгоритм Джонсона

1. Додати до графу *G* нову вершину *s*; з’єднати вершину *s* з усіма вершинами графу *G* і встановити вагу нових дуг рівною 0. Отримали трансформований граф *G’*.
2. Використати алгоритм Белмана-Форда для графу *G’*, починаючі з вершини *s* (тобто знайти найкоротші відстані від вершини *s* до решти вершин графу – значення *h*(*v*)). Якщо цей крок вкаже на присутність від’ємних циклів, то кінець роботи алгоритму.
3. Перевантажити дуги графу *G’*, використовуючи значення, які отримані з роботи алгоритму Белмана-Форда: *w*’(*u*,*v*) = *w*(*u*,*v*) + *h*(*u*) – *h*(*v*), де *w*(*u*,*v*) вага ребра у графі *G*, *w*’(*u*,*v*) – нова вага ребра, *h*(*u*) та *h*(*v*) найкоротші відстані вершини s графу *G'.*
4. Видалити вершину s з графу *G'* та застосувати до нього алгоритм Дейкстри для знаходження найкоротшого шляху між будь-якими парами вершин.