ЛЕКЦИЯ 6

1. Алгоритм Флойда-Уоршела

1.1. Описание алгоритма

Задача алгоритма: *нахождение кратчайших расстояний между всеми вершинами взвешенного ориентированного графа.*

Алгоритм: на каждом шаге проверяем новое количество вершин, т.е. на первом шаге имеем путь между двумя соседними вершинами — ребро, соединяющее вершины. На втором шаге выбираем какую-то вершину и смотрим, будет ли короче путь между двумя изначальными вершинами, если пройти через эту выбранную точку. Далее выбираем следующую точку и т.д. Таким образом мы постепенно исследуем все пути из вершины в вершину через постепенно растущее число вершин.

Строим последовательность матриц (методом динамического программирования) $a_{ij}^0 \to a_{ij}^1 \to a_{ij}^2 \to \cdots \to a_{ij}^n$. a_{ij}^k — кратчайшее расстояние от i-ой до j-ой вершины, при этом как промежуточными пользуемся от 1-ой до k-ой.
$a_{ij}^0 = W_{ij}$ — промежуточными вершинами пользоваться нельзя.
Правило поиска следующей матрицы: $a_{ij}^k = \min(a_{ij}^{k-1}, a_{ik}^{k-1} + a_{kj}^{k-1})$.
Сложность алгоритма: $O(N^3)$.

Более подробно алгоритм описан в википедии.

1.2. Реализация

Будем запоминать все предыдущие матрицы (что необязательно).

Программа №1.1. Реализация алгоритма Флойда-Уоршела

```
1    A = [[[INF]*n for i in range(n)] for k in range(n+1)] # INF - условная
     бесконечность, n - число ребер
2    for i in range(n):
3        A[0][i][:] = W[i] # При копировании весовой матрицы W расстояние от вершины
     до себя равно нулю; забиваем матрицу рёбер т.е. расстояния в начальный момент.
4    for k in range(1, n+1):
5        for i in range(n):
6            for j in range(n):
7                A[k][i][j] = min(A[k-1][i][j], A[k-1][i][k]+A[k-1][k][j]) # Добав-
     ляем путь от i до j вершины через новую вершину, если такой путь короче
```

Алгоритм не работает с циклами отрицательного веса, т.к. можно «накрутить» минус бесконечность. Но если при этом путь от i-ой до j-ой вершины не содержит такого цикла, то алгоритм сработает правильно.

2. Топологическая сортировка

Если граф не содержит циклов, то его вершины можно пронумеровать так, что любое ребро идет от вершины с меньшим номером к вершине с большим номером.

Топологическая сортировка — упорядочивание вершин бесконтурного ориентированного графа согласно частичному порядку, заданному ребрами орграфа на множестве его вершин.

Т.е. если граф не содержит циклов, то его вершины можно пронумеровать так, что любое ребро идет от вершины с меньшим номером к вершине с большим номером.
Применяется в различных прикладных задачах.

2.1. Алгоритм Кана

A(k) — множество вершин, от которых «зависит» вершина k (т.е. в которую можно прийти от других вершин).
P — последовательность вершин.
Алгоритм:
`Пока` $|P|$ `< N:`
`Найти вершину v, у которой A(v) =` $\varnothing$
`P.append(v)`
`Вычеркнуть v из всех множеств A(k).`
Алгоритм очень громоздкий и неэффективный.

2.2. Алгоритм Тарьяна

2.2.1. Алгоритм

Сложность алгоритма: $O(n)$.
По факту мы просто осуществляем обход в глубину, в котором мы будем красить вершины.
Подход алгоритма: DFS с покраской вершин (белая/серая/черная).
С любой вершины не used вершины запускаем DFS. Белые вершины — еще не тронутые, серые вершины — те, в которые алгоритм вошел, черные вершины — те, из которого алгоритм вышел. Попытка входа в серую вершины означает наличие цикла, значит алгоритм невозможен. При выходе из вершины (в момент покраски ее в черный цвет) добавляем ее в начало списка.

2.2.2. Реализация алгоритма

Программа №2.1. Реализация алгоритма Тарьяна

```
1    Visited = [False]*(n + 1)
2    Ans = []
3
4    def DFS(start):
5        Visited[start] = True
6        for u in V[start]:
7            if not Visited[u]:
8                DFS(u)
9        Ans.append(start)
10
11   for i in range(1, n + 1):
12       if not Visited(i):
13           DFS(i)
14   Ans = Ans[::-1]
```

3. Неэффективные алгоритмы

3.1. Задача Коммивояжера

Задача: *найти минимальный по весу гамильтонов цикл. Сложность алгоритма:* $O(N!)$.

В графе есть вершины. Коммивояжер захотел посетить все эти вершины и вернутся домой. Гамильтонов цикл точно есть в этой системе (это легко проверить, соединив все вершины по порядку). То есть, задача сводится к тому, чтобы найти минимальный по весу Гамильтонов цикл. Но эта задача плохо реализуется при большом числе вершин, так как основа решения — перебор. Т.е. имеем $N!$ потенциально возможных гамильтоновых циклов, каждый имеет свой вес.

К примеру, на рис. 1 изображен оптимальный маршрут коммивояжёра через 15 крупнейших городов Германии. Указанный маршрут является самым коротким из всех возможных 43 589 145 600 вариантов.

Рис. 1. Оптимальный маршрут коммивояжёра через 15 крупнейших городов Германии

Таким образом оптимизировать эту задачу невозможно.

3.2. Задача о китайском почтальоне

Задача: *пройти по каждому ребру графа минимум 1 раз, чтобы доставить почту. Требуется найти такой цикл минимального суммарного веса.*

То есть, нам необходимо найти цикл минимального суммарного веса, такой, что он проходит по ребру хотя бы 1 раз. Важное замечание — если граф Эйлеров, то Эйлеров цикл и есть решение этой задачи. А вот если его нет, то задача усложняется — решение можно найти только полным перебором. Итоговый алгоритм получается неэффективным, т.к. его асимптотика $O(N!)$.

Г. С. Демьянов, VK
С. С. Клявинек, VK