**Лекция 5. Постановка задачи о поиске кратчайшего пути на взвешенном орграфе**

**13. Кратчайшие пути**

В этом разделе будут рассмотрены алгоритмы поиска маршрутов, позволяющие не только «дойти до выбранного места», но и сделать это «самым коротким путем». Все наши задачи о нахождении кратчайшего пути между двумя фиксированными вершинами будут основываться на следующем предположении: **если кратчайший путь от *s* до *t* проходит через вершину *k*, то его отрезок от *s* до *k* будет кратчайшим путем от *s* до *k* и отрезок от *k* до *t* будет кратчайшим путем от *k* до *t*.**

Существует класс задач, для которых это не так. Например, это задачи с фиксированными платежами, когда за прохождение вершины сети взимается плата. В общем случае эти задачи возникают в **системах с последействием** – для нахождения оптимального продолжения решения мало знать текущее состояние задачи, нужно помнить историю, как мы туда попали. Решаются такие задачи довольно сложно, в данном пособии рассматриваться не будут.

Дадим основные определения и обозначения для объектов, из которых будет построена наша математическая модель.

Будем рассматривать ориентированные графы, дугам которых приписаны веса (рис. 13.1).

Это означает, что каждой дуге (*u*→*v*) поставлено в соответствие вещественное число A[*u*, *v*]. Матрицу весов дуг данного графа будем обозначать *A*, несуществующим дугам будем приписывать веса, равные +∞, т.е. A[*u*, *v*] = +∞, если дуга (*u*→*v*) не существует.

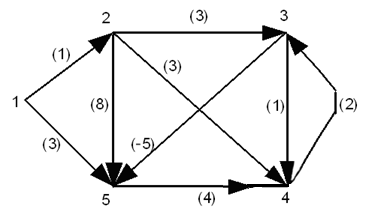


Рис. 13.1. Модельный граф

Матрица весов *A* для модельного графа (рис. 13.1):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| +∞ | 1 | +∞ | +∞ | 3 |
| +∞ | +∞ | 3 | 3 | 8 |
| +∞ | +∞ | +∞ | 1 | –5 |
| +∞ | +∞ | 2 | +∞ | +∞ |
| +∞ | +∞ | +∞ | 4 | +∞ |

В программе граф будет задаваться матрицей весов и (если потребуется) списками инцидентности PRED и SLED. В списках PRED для каждой вершины *v* хранится список ее предшественниц *u*, т.е. вершин, из которых дуга идет в эту вершину.

В списках SLED для каждой вершины *v* хранится список ее потомков, т.е. вершин, в которые из *v* идет дуга.

Длина пути будет подсчитываться как сумма весов, входящих в него дуг.

Зафиксируем в графе пару вершин *s* (источник) и *t* (сток).

Длину кратчайшего пути от *s* до *t* будем обозначать DD[*s*, *t*], где *DD* – матрица расстояний между всеми парами вершин. Если такого пути не существует, то будем считать, что DD[*s*, *t*] = +∞.

Если источник *s* зафиксирован, то расстояния от *s* до остальных вершин графа будут храниться в векторе (одномерном массиве) *D*, где D[*v*] – расстояние от *s* до *v*.

Как видно из определения, дугам могут быть приписаны отрицательные веса. Если в графе существует контур (замкнутый путь) отрицательной длины, то понятие кратчайшего пути теряет смысл: обходя этот контур сколь угодно большое число раз, можно сделать длину пути как угодно малой.

Если потребовать, чтобы в качестве кратчайшего пути искать только элементарные пути, т.е. пути, не содержащие контуров, то задача станет вполне корректной, но, увы, **NP**–**полной**. Поскольку мы хотим разработать для задачи эффективные алгоритмы, придется умерить свои требования и поставить задачу так, чтобы мы могли ее решить за полиномиальное время.

Окончательная постановка: **будем искать кратчайшие пути на взвешенных орграфах без контуров отрицательного веса**.

На таких графах кратчайшие пути автоматически будут элементарными, так как добавление контура может привести только к увеличению стоимости пути. В этом разделе мы рассмотрим четыре алгоритма, которые будут вычислять матрицу *D* расстояний между источником и всеми остальными вершинами:

1. первый подходит графам самого общего вида, т.е. без контуров отрицательной длины;
2. второй применяется для графов с неотрицательными весами;
3. третий – для бесконтурных графов;
4. четвертый на графе общего вида находит расстояния между всеми парами вершин.

Сразу возникает вопрос: зачем нужны четыре различных алгоритма, когда для решения всех этих задач было бы достаточно одного, самого первого?

Дело в том, что идет борьба за эффективность, т.е. вычислительную сложность алгоритма. Чем более специального вида граф, тем более быстрым специализированным алгоритмом решается задача о кратчайшем пути.

Результатом работы любого из этих алгоритмов являются не сами пути, а матрицы расстояний. Поэтому решим следующую задачу: **зная расстояния от источника *s* до всех остальных вершин, найдем путь от источника до какой-то выбранной вершины *t*.**

Если нам удастся по последней вершине *t* восстановить предпоследнюю (назовем ее *v*), то задача решена. Восстановим предпоследнюю, по ней предпредпоследнюю и так до тех пор, пока не дойдем до источника.

Предпоследняя вершина *v* кратчайшего пути обладает следующим свойством: D[*s*, *t*] = = D[*s*, *v*] + A[*v*, *t*].

**АЛГОРИТМ 13.1** *{Восстановление кратчайшего пути от s до t.}*

*Данные:* Орграф *G = <V, E*>, заданный списками PRED[*v*], вектор расстояний D[*v*], выделенные вершины *s* и *t*, матрица весов A[*u*,*v*], *u*, *v* ∈ *V*.

*Результаты:* СТЕК, содержащий путь от *s* до *t*.

1 begin

2 СТЕК:= 0; СТЕК ⇐ t; v:= t;

3 while v<>s do

4 begin

5 u:= первая из списка PRED[v];

6 while D[v]<>D[u]+A[u,v] do

7 u := следующая из списка PRED[v];

8 СТЕК ⇐ u; v:= u;

9 end

10 end.

**Вычислительная сложность алгоритма**

Так как для поиска всех вершин *u* происходит просмотр не более *m* дуг (в списках PRED), то вычислительная сложность этого алгоритма О(*m*).

**Замечание.** При наличии в графе **контура нулевого веса** эта процедура может зациклиться, и ее следует модифицировать.

***Вопрос.*** Что произойдёт при вызове этой процедуры в случае, если путь от *s* до *t* не существует?

***Ответ*.** Цикл (6–7) не найдёт предыдущую вершину, не выполнится условие выхода из цикла.