Некоторые алгоритмы на графах – II

Изучим задачу об отыскании кратчайшего пути. На практике она может возникнуть, например, из следующих соображений: имеется карта автомобильных дорог, для каждой дороги известна ее длина (в километрах). Нужно найти кратчайший путь из одного города в другой. Еще один вариант: известна стоимость проезда из одного города в другой, но не между всеми городами существуют рейсы. Требуется составить маршрут таким образом, чтобы стоимость проезда между заданными городами была минимальна.

Математическая постановка задачи выглядит следующим образом. Дан ориентированный взвешенный граф $G = (V, E)$ с вещественной весовой функцией. Весом пути называется сумма весов ребер, входящих в этом путь. Необходимо найти такой путь между двумя заданными вершинами, чтобы его вес был минимально возможным (если путь между вершинами вообще существует).

Впрочем, отыскание пути из одной заданной вершины в другую заданную вершину – это не единственно возможная постановка задачи. Может потребоваться отыскать кратчайшие пути из данной вершины до всех остальных, кратчайшие пути до данной вершины из всех остальных (в некотором смысле эти две задачи взаимно обратные), или даже найти кратчайшие пути для всех пар вершин.

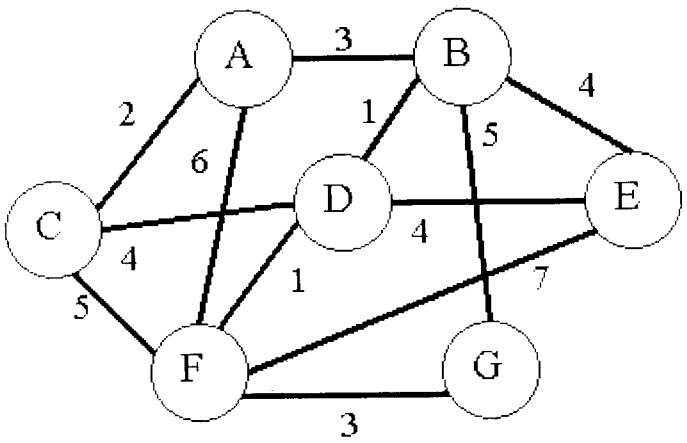
В большинстве (практически значимых) случаев веса ребер считаются неотрицательными, и некоторые алгоритмы используют этот факт. Однако в некоторых приложениях веса ребер могут быть отрицательными, и тогда становится важным, существуют ли циклы отрицательного веса. Действительно, если добраться до такого цикла, то можно будет обходить его сколько угодно раз, уменьшая и уменьшая вес кратчайшего пути. В таком случае можно считать, что вес кратчайшего пути – это $-\infty$. Если же циклов отрицательного веса нет, то любой цикл можно исключить из рассмотрения, не удлиняя пути. Путей без циклов конечное число, и вес кратчайшего пути будет корректно определен.

Первый алгоритм, который будет изучен – это алгоритм Дейкстры. Он отыскивает кратчайшие пути из заданной вершины до всех остальных при условии, что все ребра имеют неотрицательные веса. Этот алгоритм относится к классу алгоритмов динамического программирования (см. занятие 28), поскольку (при данных условиях) любая часть кратчайшего пути сама есть кратчайший путь. Идея его состоит в следующем. Для каждой вершины v будем хранить верхнюю оценку веса кратчайшего пути (обозначим ее $d[v]$) из s (стартовой вершины) в v, которую постепенно будем уточнять. Уточнение будет происходить так: если найдется вершина u, для которой выполняется $d[v] > d[u] + w(u, v)$, где $w(u, v)$ – вес ребра (u, v), то оценка меняется. Отметим, что эта техника будет так или иначе использоваться и в других алгоритмах отыскания кратчайших путей.

В процессе работы алгоритма все вершины графа разбиваются на два множества: те, для которых оценка $d[v]$ уже вычислена, и те, для которых она еще неизвестна. Изначально в первое множество включается только сама стартовая вершина (для нее, очевидно, $d[s] = 0$), для остальных же вершин можно полагать $d[v] = \infty$. Далее первое множество вершин расширяется за счет выбора из второго множества очередной вершины, для которой оценка минимальна. После того, как эта вершина будет помещена в первое множество, нужно будет скорректировать информацию для всех вершин второго множества. Отметим, что наименьшая оценка будет достигаться для вершин из второго множества на пути, который проходит полностью через вершины первого множества (и единственной вершиной из второго множества будет конечная вершина этого пути).

Рассмотрим, как работает этот алгоритм, на примере графа, изображенного на рис. 1. Пусть нам нужно найти кратчайшие пути до всех вершин из вершины A. На первом шаге, как уже было сказано, помещаем вершину A в первое множество и полагаем $d[A] = 0$, все остальные вершины пока находятся во втором множестве. Теперь мы должны выбрать из этого второго множества ту вершину, к которой ведет наиболее короткий (т.е. меньшего веса) путь из A.

Как видно из рисунка, вершина A соединена путями с B (длины 3), C (длины 2) и F (длины 6). Поэтому первое множество следует дополнить вершиной C. Теперь нужно учесть, что вершина C связана с вершинами D и F, поэтому можно сделать для них оценку. Вершина D не связана с A непосредственно, и поэтому (пока) более короткого пути в нее из A, нежели ACD, просто не существует. А что касается вершины F, то прямой путь до нее имеет длину 6, а путь ACF имеет длину 7. Поэтому оценочным значением будет длина прямого пути. Будем заносить полученные оценочные значения в таблицу, выделяя жирным шрифтом обновленные значения. В графе «Первое множество» будем записывать вершины, которые составляют это множество по итогам очередной итерации (скорректированные оценки вынесены в следующую итерацию).



Отыскиваем следующую вершину для включения ее в первое множество – это будет вершина B. Вершина B связана с вершинами D, E, G. Путь ABD имеет длину 3+1 = 4, что меньше, чем ACD (2+4 = 6), поэтому для вершины D следует изменить оценку на 4. Что касается вершин E и G, то для них оценка выполняется впервые и она заведомо меньше ∞: путь ABE имеет длину 7, а путь ABG – длину 8.

№№	d[A]	d[B]	d[C]	d[D]	d[E]	d[F]	d[G]	Первое множество
0	0	∞	∞	∞	∞	∞	∞	A
1	0	**3**	**2**	∞	∞	**6**	∞	A, C
2	0	3	2	**6**	∞	6	∞	A, C, B
3	0	3	2	**4**	**7**	6	**8**	A, C, B, D
4	0	3	2	4	7	**5**	8	A, C, B, D, F
5	0	3	2	4	7	5	8	A, C, B, D, F, E
6	0	3	2	4	7	5	8	A, C, B, D, F, E, G

Теперь, глядя на строчку таблицы для итерации 3, можно заключить, что очередной вершиной, включаемой в первое множество, будет вершина D. Обновления могут потребовать оценки для вершин F и E (кроме них вершина D соединена ребрами с B и C, но они уже входят в первое множество). Как было установлено, кратчайший путь от A до D имеет длину 4, путь же от D до F имеет длину 1, что дает в сумме 5. Существующая оценка пути до F равна 6 и должна быть заменена. Длина пути от D до E составляет 4, что в сумме с уже известным путем дает 8. Существующая оценка 7 не требует изменений. По итогам итерации следующей добавляемой в первое множество вершиной будет вершина F.

Вершина F соединена ребрами с обеими вершинами, которые пока еще пребывают во втором множестве – и с E, и с G. Ребро FE имеет длину 7, а ребро FG – длину 3. В сумме с существующей длиной кратчайшего пути от A до F (5) получаем длины не меньшие, нежели существующие оценки. Поэтому обновления информации не происходит, а вершина E включается в первое множество. Поскольку она с вершиной G не соединена непосредственно, обновления информации для G не произойдет, и она просто будет добавлена в первое множество на последней итерации. Второе множество вершин после этого окажется пустым, а все кратчайшие пути из вершины A до других вершин данного графа будут найдены.

Задача. Напишите реализацию алгоритма Дейкстры.

Решение.

Приведем процедуру на псевдокоде (величина w[u,v] означает вес пути от u к v, G – граф, s – стартовая вершина).

```
procedure d_alg (G, s);
var Second: множество вершин;
    v, u: вершины;
begin
  Second := [];
  for каждая вершина v графа do
  begin
    d[v] := ∞;
    Second := Second + [v];
  end;
  d[s] := 0;
  while Second <> [] do
  begin
    u : = вершина, для которой достигается минимум d[u];
    Second := Second - [u];
    for каждая вершина v из множества Second do
      d[v] := min(d[v], d[u]+w[u, v]);
  end;
end;
```

Задачи.

1. Перепишите реализацию алгоритма Дейкстры на Pascal, описав необходимые структуры данных (какие именно – выберите сами). Оцените время работы алгоритма.
2. Напишите программу, которая находит кратчайшие пути между всеми парами вершин в графе. Считайте, что циклы с отрицательной стоимостью в графе отсутствуют (но ребра с отрицательным весом допустимы). Оцените время работы алгоритма.

Комментарий (к задаче 2).

Алгоритм, отыскивающий кратчайшие пути для всех пар вершин ориентированного взвешенного графа, был разработан Флойдом и Уоршоллом. Он также основан на технике динамического программирования.

Алгоритм Дейкстры выделял последнее ребро пути. Алгоритм Флойда – Уоршолла действует иначе: для него важно, какие вершины будут использованы в качестве промежуточных, а характеристикой пути является максимальный номер промежуточной вершины в нем (нумерация вершин фиксируется заранее).

Обозначим через S(i, j, k) наименьшую длину маршрута из i в j, если в качестве промежуточных пунктов разрешено использовать вершины с номерами, не превосходящими k. Тогда можно построить следующую итеративную систему. Сначала полагаем S(i, j, 0) = w[i,j] (весу ребра, соединяющего вершины i и j; если оно не существует, можно считать, что это бесконечность). Далее добавляем к «разрешенным» в качестве промежуточных пунктов очередную вершину и корректируем при необходимости оценку минимальной длины пути:

$$S(i, j, k+1) = \min (S(i, j, k), S(i, k+1, k) + S(k+1, j, k)).$$

Первая величина соответствует уже существующей оценке, вторая – случаю, когда пункт k+1 используется в качестве промежуточного.

Попробуйте выполнить этот алгоритм для графа, изображенного на рис. 1., вручную.

Домашнее задание

1. Напишите программу, которая находит и выводит кратчайшие пути между всеми парами вершин в графе (алгоритм Флойда – Уоршолла).
2. Напишите программу, которая будет отыскивать в графе циклы с отрицательным весом.

Указание. Воспользуйтесь алгоритмом Флойда-Уоршолла.

3. Пусть имеется n городов, пронумерованных числами от 1 до n. Для каждой пары городов (i,j) хранится целое число a[i, j] – цена билета. Рейсы существуют между любыми городами, a[i,i] = 0, a[i, j] может отличаться от a[j, i]. Наименьшей стоимостью проезда из i в j считается минимально возможная сумма цен билетов для маршрутов (в т.ч. с пересадка-

Нахождение кратчайших путей между парами вершин графа.

Предположим, что мы имеем помеченный орграф, который содержит время полета по маршрутам, связывающим определенные города, и мы хоти построить таблицу, где приводилось бы минимальное время перелета из одного (произвольного) города в любой другой. В этом случае мы сталкиваемся с общей задачей нахождения кратчайших путей, то есть нахождения кратчайших путей между всеми парами вершин орграфа. Более строгая формулировка этой задачи следующая: есть ориентированный граф G=(V,E), каждой дуге $v \to w$ этого графа сопоставлена неотрицательная стоимость C[v,w]. Общая задача нахождения кратчайших путей заключается в нахождении для каждой упорядоченной пары вершин (v,w) любого пути от вершины v в вершины w, длина которого минимальна среди всех возможных путей от v к w.

Можно решить эту задачу, последовательно применяя алгоритм Дейкстры для каждой вершины, объявляемой в качестве источника. Но существует прямой способ решения данной задачи, использующий алгоритм Флойда. Для определенности положим, что вершины графа последовательно пронумерованы от 1 до n. Алгоритм Флойда использует матрицу A размера n x n, в которой вычисляются длины кратчайших путей. Вначале A[i,j]=C[i,j] для всех $i \neq j$. Если дуга $i \to j$ отсутствует, то $C[i,j] = \infty$. Каждый диагональный элемент матрицы A равен 0.

Над матрицей A выполняется n итераций. После k-й итерации A[i,j] содержит значение наименьшей длины путей из вершины i в вершину j, которые не проходят через вершины с номером, большим k. Другими словами, между концевыми вершинами пути i и j могут находиться только вершины, номера которых меньше или равны k.

На k-й итерации для вычисления матрицы A применяется следующая формула:
$A_k[i,j]=\min(A_{k-1}[i,j], A_{k-1}[i,k]+A_{k-1}[k,i])$.

Нижний индекс k обозначает значение матрицы A после k-й итерации, но это не означает, что существует n различных матриц, этот индекс используется для сокращения записи. Графическая интерпретация этой формулы показана на рисунке:

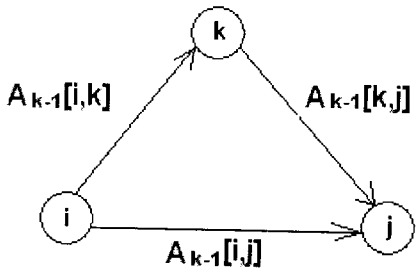


Рис.1 Включение вершины k в путь от вершины i к вершинеj

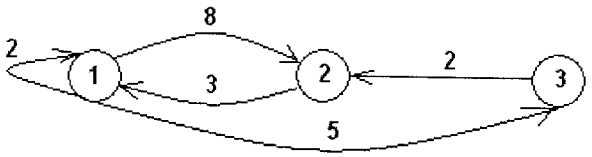


Рис.2 Помеченный ориентированный граф.

Для вычисления $A_k[i,j]$ проводится сравнение величины $A_{k-1}[i,j]$ (то есть стоимость пути от вершины i к вершине j без участия вершины k или другой вершины с более высоким номером) с величиной $A_{k-1}[i,k]+A_{k-1}[k,j]$ (стоимость пути от величины i до вершины k плюс стоимость пути от вершины k до вершины j). Если путь через вершину k дешевле, чем $A_{k-1}[i,j]$, то величина $A_k[i,j]$ изменяется.

На предыдущем рисунке показан помеченный граф, а на следующем рисунке – значения матрицы A после трех итераций.

	1	2	3
1	0	8	5
2	3	0	∞
3	∞	2	0

$A_0[i,j]$

	1	2	3
1	0	8	5
2	3	0	8
3	∞	2	0

$A_1[i,j]$

	1	2	3
1	0	8	5
2	3	0	8
3	5	2	0

$A_2[i,j]$

	1	2	3
1	0	7	5
2	3	0	8
3	5	2	0

$A_3[i,j]$

Равенства $A_k[i,k]=A_{k-1}[i,k]$ и $A_k[k,j]=A_{k-1}[k,j]$ означают, что на k-й итерации элементы матрицы A, стоящие в k-м столбце, не изменяются. Более того, все вычисления можно выполнить с

применением только одной копии матрицы A. Процедура, реализующая алгоритм Флойда, выглядит следующим образом:

```
Procedure Floyd (var A: array[1..n,1..n] of real; C: array[1..n,1..n] of real);
        var  i,j,k:integer;
        begin
        for i:=1 to n do
          for j:=1 to n do
            A[i,j]:=C[i,j];
        for i:=1 to n do
            A[i,i]:=0;
        for k:=1 to n do
          for i:=1 to n do
            for j:=1 to n do
                  if  A[i,k]+ A[k,j]< A[i,j] then A[i,j]:= A[i,k]+ A[k,j];
        end;
```

Вывод на печать кратчайших путей.

Во многих ситуациях требуется распечатать самый дешевый путь от одной вершины к другой. Чтобы восстановить при необходимости кратчайшие пути, можно в алгоритме Флойда ввести еще одну матрицу P, в которой элемент P[i,j] содержит вершину k, полученную при нахождении наименьшего значения A[i,j]. Если P[i,j]=0, то кратчайший путь из вершины i в вершину j состоит из одной дуги $i \rightarrow j$. Модифицированная версия алгоритма Флойда, позволяющая восстанавливать кратчайшие пути, выглядит следующим образом:

```
Procedure Floyd (var A: array[1..n,1..n] of real; C: array[1..n,1..n] of real; P: array[1..n,1..n] of real);
        var  i,j,k:integer;
        begin
        for i:=1 to n do
          for j:=1 to n do
            begin
             A[i,j]:=C[i,j];
             P[i,j]:=0;
            end;
        for i:=1 to n do
            A[i,i]:=0;
        for k:=1 to n do
          for i:=1 to n do
            for j:=1 to n do
                  if  A[i,k]+ A[k,j]< A[i,j] then begin A[i,j]:= A[i,k]+ A[k,j]; P[i,j]:=k; end;
        end;
```

Для вывода на печать последовательности вершин, составляющих кратчайший путь от вершины i до вершины j, вызывается процедура path(i,j), код которой приведен ниже:

```
Procedure path(i,j:integer);
var  k: integer;
begin
        k:= P[i,j];
        if k=0 then return;
        path(i,k);
        writeln(k);
        path(k,j);
end;
```

Результирующая матрица P для орграфа из второго рисунка выглядит следующим образом:

	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0

P