

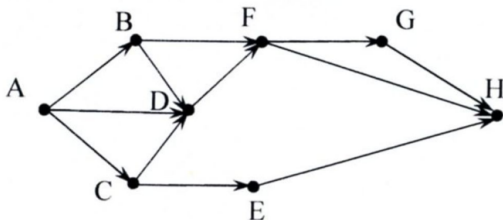
Графы

Юрий Литвинов
yurii.litvinov@gmail.com

19.11.2019

Что это и зачем

- ▶ Граф — совокупность вершин и рёбер
- ▶ Нужен для моделирования систем с нетривиальными связями между элементами
- ▶ Например:
 - ▶ Граф дорог, соединяющих города
 - ▶ Граф компьютерной сети
 - ▶ Граф зависимостей модулей в процессе компиляции
 - ▶ Дерево — подвид графа
- ▶ Есть куча известных алгоритмов на графах, позволяющих узнать о них очень многое



Определения

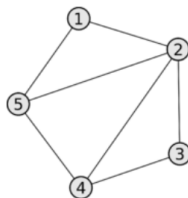
- ▶ Ориентированный граф G — пара из множества вершин V и множества дуг (или рёбер) E , где E — упорядоченная пара вершин (v, w)
 - ▶ E — бинарное отношение над множеством вершин
 - ▶ v называется началом, w — концом дуги
 - ▶ Дуги вида (v, v) называются петлями
- ▶ Путём в орграфе называется последовательность вершин, для которых существуют дуги из предыдущей в следующую
 - ▶ Длина пути — количество дуг, составляющих путь
 - ▶ Путь называется простым, если все вершины на нём, за исключением, быть может, первой и последней, различны
 - ▶ Цикл — это простой путь длины не менее 1, который начинается и заканчивается в одной вершине
 - ▶ Граф без циклов называется ациклическим

Ещё определения

- ▶ Неориентированный граф — это ориентированный граф, у которого для каждого ребра (v, w) существует противоположное ребро (w, v)
 - ▶ то есть отношение E симметрично
- ▶ Если $e = (u, v) \in E$, то вершины u и v называются смежными в G , а ребро e и эти вершины называются инцидентными
- ▶ Степенью вершины в неориентированном графе называется число смежных с ней вершин
 - ▶ Вершина степени 0 называется изолированной
- ▶ Граф может быть помеченным (или взвешенным) – каждой вершине и/или дуге сопоставлена некоторая метка (число, строка и т.д.)

Матрица смежности

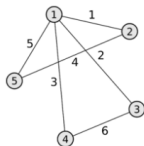
- ▶ Для графа из n вершин матрица смежности — матрица размера n на n , где в строке i и столбце j стоит 1 (или true), если есть дуга из i в j
 - ▶ Для неориентированного графа матрица симметрична относительно главной диагонали
 - ▶ Для взвешенного графа вместо 1 стоит вес дуги
- ▶ Занимает $O(n^2)$ памяти, проверка наличия дуги и определение веса — за константное время



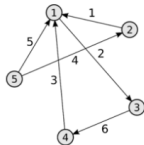
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Матрица инцидентности

- ▶ Для графа из n вершин и m ребёр матрица инцидентности — матрица размера m на n , где строка соответствует вершине, а столбец ребру
 - ▶ Для ориентированного графа в ячейке (i, j) стоит 1, если ребро i выходит из вершины j , и -1, если входит
 - ▶ Для неориентированного графа в ячейке (i, j) стоит 1, если ребро i инцидентно вершине j
 - ▶ Нужно специальное значение для петель



$$\begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$



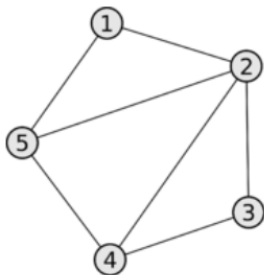
$$\begin{pmatrix} -1 & 1 & -1 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Матрица инцидентности, свойства

- ▶ В каждом столбце только два элемента ненулевые (1, если петля)
- ▶ Для неориентированного графа сумма элементов в столбце равна 2, в строке — степени вершины
- ▶ Требуется $O(m * n)$ памяти для хранения
- ▶ Проверка наличия ребра между двумя вершинами — за $O(m)$
- ▶ Поиск инцидентных ребер вершин — за $O(n)$, проверка на инцидентность — за $O(1)$

Список смежности

- ▶ Для каждой вершины хранится список вершин, смежных с данной
 - ▶ Как правило, списки лежат в массиве длины n
 - ▶ Вместе с номером смежной вершины в списке может лежать вес ребра
- ▶ Требует $O(m + n)$ памяти для хранения
- ▶ Проверка наличия ребра между двумя вершинами — $O(m)$
 - ▶ Если в графе мало рёбер, в среднем $O(1)$
- ▶ Несколько сложнее в реализации



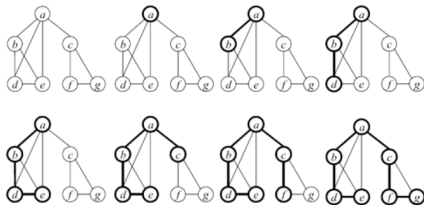
1: 2, 5
2: 1, 5, 4, 3
3: 2, 4
4: 5, 2, 3
5: 1, 2, 4

Достижимость

- ▶ Вершина w называется достижимой из вершины v , если $v = w$ или в G есть путь от v в w
 - ▶ Рефлексивное и транзитивное замыкание отношения E
 - ▶ Если E симметрично, то достижимость — отношение эквивалентности
 - ▶ Классы эквивалентности по отношению достижимости называются компонентами связности
 - ▶ Для ориентированных графов эквивалентность — отношение взаимной достижимости
- ▶ Проверка на достижимость — обходы графа в глубину или ширину, алгоритм Уоршелла

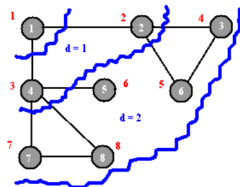
Обход в глубину

- ▶ Посещаем вершину
- ▶ Рекурсивно обходим все смежные ей вершины, в которых мы ещё не были
 - ▶ Нужно множество (или битовая шкала) посещённых вершин, проще всего передавать как параметр в рекурсивный вызов
- ▶ Можно нерекурсивно, тогда нам потребуется стек рассматриваемых вершин
 - ▶ И множество посещённых



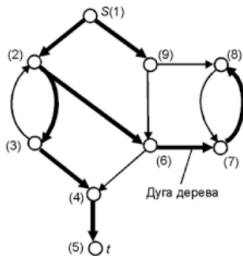
Обход в ширину

- ▶ То же, что и обход в глубину, но вместо стека рассматриваемых вершин — очередь
- ▶ На каждой итерации берём из очереди вершину, посещаем её и кладём в очередь все смежные вершины, в которых мы ещё не были
 - ▶ Тоже требуется множество непосещённых вершин
- ▶ Если знаем, что цель недалеко, обход в ширину эффективнее
- ▶ Обход в глубину зато строит глубинное остовное дерево (на самом деле, глубинный остовный лес)



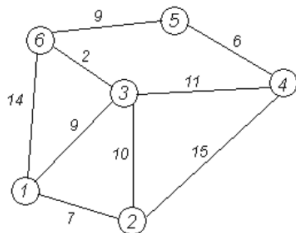
Проверка графа на ацикличность

- ▶ Остовное дерево графа — подграф, содержащий все вершины графа и являющийся деревом
 - ▶ То есть не содержащий циклов, даже если рассматривать граф как неориентированный
- ▶ Глубинное остовное дерево — путь алгоритма поиска в глубину по графу
- ▶ Обратная дуга — дуга, не принадлежащая основному дереву, ведущая от потомка к предку
 - ▶ Если обратных дуг нет, в графе нет циклов
- ▶ Посещая вершину, красим её в серый, выходя из её поддерева — в чёрный
- ▶ Если дуга ведёт в серую вершину, мы нашли цикл



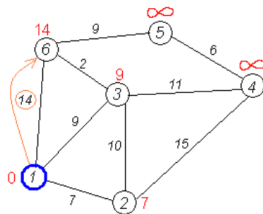
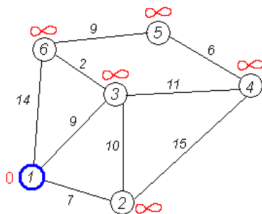
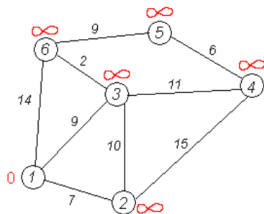
Задача поиска кратчайшего пути в графе

- ▶ Дан взвешенный ориентированный граф $G = (V, E)$ с неотрицательными весами дуг
- ▶ Одна вершина $s \in V$ помечена как стартовая
- ▶ Задача — найти кратчайшие пути от стартовой вершины до всех остальных вершин
 - ▶ Длина пути определяется как сумма весов дуг, составляющих путь
- ▶ Парадокс изобретателя — иногда проще решить более общую задачу и получить из неё решение частной задачи, чем решать частную
 - ▶ Знание целевой вершины поиска кратчайшего пути не упрощает



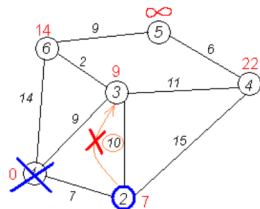
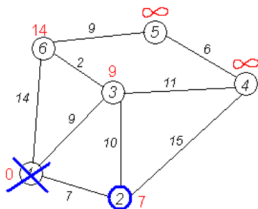
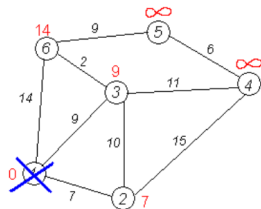
Алгоритм Дейкстры

Поиск в ширину, где мы запоминаем длину кратчайшего пути в посещённой вершине, обновляя её, если нашли лучше



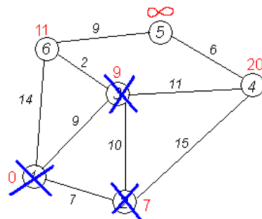
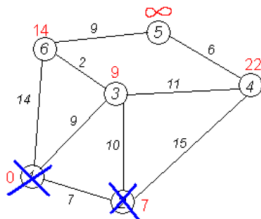
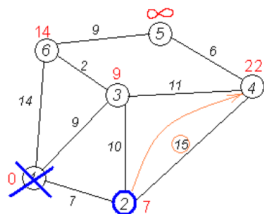
Алгоритм Дейкстры, пример, продолжение

Когда все соседи вершины посещены, посчитанная для неё длина пути окончательно и минимальна, выкидываем её из рассмотрения



Алгоритм Дейкстры, пример, продолжение (2)

Продолжаем, пока не посетили все вершины (правда, бывают несвязные графы)



Алгоритм Дейкстры, псевдокод

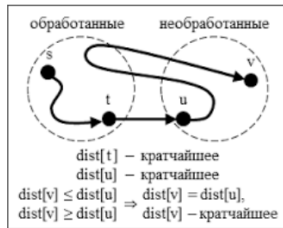
```
1: procedure Dijkstra(s)
2:   for  $v \in V$  do
3:      $d[v] \leftarrow \infty$ 
4:      $used[v] \leftarrow false$ 
5:   end for
6:    $d[s] \leftarrow 0$ 
7:   for  $i \in V$  do
8:      $v \leftarrow null$ 
9:     for  $j \in V$  do
10:      if  $!used[j]$  and  $(v == null \text{ or } d[j] < d[v])$  then
11:         $v \leftarrow j$ 
12:      end if
13:    end for
14:    if  $d[v] == \infty$  then
15:      break
16:    end if
17:     $used[v] \leftarrow true$ 
18:    for  $e \in$  исходящие из  $v$  рёбра do
19:      if  $d[v] + e.len < d[e.to]$  then
20:         $d[e.to] \leftarrow d[v] + e.len$ 
21:      end if
22:    end for
23:  end for
24: end procedure
```

Тонкости реализации

- ▶ Иногда надо знать не длину пути, а сам путь
 - ▶ Тогда помимо массива длин d полезно иметь массив родителей p , такой что $p[v] = u$, если мы пришли кратчайшим путём в v из u
 - ▶ Обновляем этот массив там же, где мы обновляем d
- ▶ Множество $used$ можно представлять битовой шкалой, или, если вершин много, множеством на двоичных деревьях или хеш-таблицах
- ▶ Поиск ближайшей вершины из множества нерассмотренных можно хранить в очереди с приоритетами, тогда просматривать всё V будет не надо (можно обойтись $O(\log(n))$)
 - ▶ Как — кучей (вспомните heapsort)
- ▶ Трудоёмкость наивной реализации — $O(n^2 + m)$, с кучей — $O(n * \log(n) + m * \log(n))$

Почему работает

- ▶ По индукции
 - ▶ На первом шаге $d(s) = 0$, расстояние до неё кратчайшее
 - ▶ Для n шагов всё ок, на $n + 1$ -м выбрана для добавления вершина v . Покажем, что посчитанный для неё сейчас путь — кратчайший.
 - ▶ Если это не так, существует непросмотренная вершина u на кратчайшем пути, $d(v) > d(u)$, потому что веса путей неотрицательны
 - ▶ Но тогда u была бы добавлена раньше, поскольку на каждом шагу выбирается ближайшая к стартовой из непросмотренных вершин
- ▶ Пример жадного алгоритма



Алгоритм Флойда

- ▶ Считает кратчайший путь из каждой вершины в каждую
- ▶ Над матрицей смежности выполняется n итераций, на 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 :

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
       $W[i, j] = \min(W[i, j], W[i, k] + W[k, j])$ 
```

- ▶ Трудоёмкость — $O(n^3)$
 - ▶ Зато очень просто реализуется
- ▶ Сам кратчайший путь — заводим матрицу P , такую, что на каждой итерации $P[i, j] = k$, дальше рекурсивно восстанавливаем путь по $P[i, k]$ и $P[k, j]$

Алгоритм Уоршелла

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      A[i, j] = A[i, j] or (A[i, k] and A[k, j])
```

- ▶ Чаще эти два алгоритма называют алгоритмом Флойда-Уоршелла, хотя они разработаны независимо (причём, за 3 года и до Флойда, и до Уоршелла, Бернардом Роем)
- ▶ Строит транзитивное замыкание отношения E
 - ▶ Легко искать компоненты связности неориентированного графа

Графы и математика

- ▶ Любой граф задаёт бинарное отношение над некоторым множеством
 - ▶ Ну он и есть бинарное отношение E , да
- ▶ Ациклический орграф задаёт отношение строгого частичного порядка:
 - ▶ Антирефлексивность: $\forall x \neg xRx$
 - ▶ Транзитивность: $\forall x, y, z : xRy \cap yRz \rightarrow xRz$
 - ▶ Пример: отношения “больше” и “меньше” на множестве вещественных чисел, отношение строгого включения множеств
- ▶ Топологическая сортировка множества V относительно отношения частичного порядка R — построение последовательности v_1, v_2, \dots, v_n : $\forall i, j \in 1..n, v_i, v_j \in V$ и v_iRv_j или $(v_i, v_j) \notin R$
 - ▶ Например, сортировка графа зависимостей файлов при сборке
 - ▶ Делается поиском в глубину (глубинная остовная нумерация)

Доклады

1. Алгоритмы Прима и Краскала
2. Алгоритм Кнута-Морриса-Пратта
3. Алгоритм Рабина-Карпа
4. Алгоритм Бойера-Мура
5. Алгоритм A^*