Алгоритмы на графах. Обходы графов. Кратчайшие пути. Остовные деревья

Определения

Ориентированный граф (сокращенно *орграф*) G = (V, E) состоит из множества вершин V и множества дуг E. Вершины также называют *узлами*, а дуги – *ориентированными ребрами.* Дуга представима в виде упорядоченной пары вершин (v, w), где вершина v называется началом, а w – концом дуги.

Неориентированный граф G = (V, E) состоит из конечного множества вершин V и множества ребер E. В отличие от ориентированного графа, здесь каждое ребро (v, w) соответствует *неупорядоченной* паре вершин: если (v, w) – неориентированное ребро, то (v, w) = (w, v).

Обходы графов

Поиск в ширину (breadth-first search)

Пусть задан граф G = (V, E) и фиксирована начальная вершина s. Алгоритм поиска в ширину перечисляет все достижимые из s (если идти по ребрам) вершины в порядке возрастания расстояния от s. Расстоянием считается длина (число ребер) кратчайшего пути. В процессе поиска из графа выделяется часть, называемая “деревом поиска в ширину” с корнем s. Она содержит все достижимые из s вершины (и только их). Для каждой из них путь из корня в дереве поиска будет одним из кратчайших путей (из начальной вершины) в графе. Алгоритм применим и к ориентированным, и к неориентированным графам.

Будем считать, что в процессе работы алгоритма вершины могут быть белыми, серыми и черными. Вначале они все белые, но в ходе работы алгоритма белая вершина может стать серой, а серая – черной (но не наоборот). Повстречав новую вершину, алгоритм красит её, так что окрашенные вершины – это в точности те, которые уже обнаружены.

Алгоритм.

d[u] – расстояние по ребрам от s до u

Q - очередь

```
BFS(G, s)
 for (для) каждой вершины u из V[G] – {S}
     do color[u] = БЕЛЫЙ
        d[u] = ∞
        родитель[u] = NIL
 color[s] = СЕРЫЙ
 d[s] = 0
 родитель[s] = NIL
 Q ← s
 while (очередь Q не пуста)
     do u ← извлечь вершину из очереди Q
          for (для) всех v смежных с u
               do if color[v] == БЕЛЫЙ
                     then color[v] = СЕРЫЙ
                          d[v] = d[u] + 1
                          родитель[v] = u
                          положить v в очередь
          color[u] = ЧЕРНЫЙ
```

Поиск в глубину

Стратегия поиска в глубину такова: идти “вглубь”, пока это возможно (есть не пройденные исходящие ребра), и возвращаться и искать другой путь, когда таких ребер нет. Так делается, пока не обнаружены все вершины, достижимые из исходной. (Если после этого остаются необнаруженные вершины, можно выбрать одну из них и повторять процесс, и делать так до тех пор, пока мы не обнаружим все вершины графа.)

Алгоритм.

`d[u]` – время начала обработки вершины u

`f[u]` – время окончания обработки вершины u

`time` – глобальное время

```
DFS(G)
 for (для) всех вершин u из V[G]
     do color[u] = БЕЛЫЙ
        родитель[u] = NIL
 time = 0
 for (для) всех вершин u из V[G]
     do if color[u] = БЕЛЫЙ
          then DFS-Visit(u)

DFS-Visit(u)
 color[u] = СЕРЫЙ
 time++
 d[u] = time
 for (для) всех v смежных с u
     do if color[v] = БЕЛЫЙ
          then родитель[v] = u
               DFS-Visit(v)
 color[u] = ЧЕРНЫЙ
 time++
 f[u] = time
```

Где используется:

- алгоритм нахождения компонент связности, мостов и точек сочленения (для этого используются времена начала и окончания обработки)
- алгоритм топологической сортировки DAG’a (directed acyclic graph)
- проверка графа на ацикличность

Кратчайшие пути

Кратчайший путь из u в v – это любой путь p из u в v, для которого $w(p) = \delta(u, v)$, где:

- $w(p)$ = сумма весов всех ребер пути p
- $\delta(u, v) = \min\{w(p)$: по всем путям p из u в v$\}$, если существует путь из u в v; ∞ - в противном случае

Алгоритм Дейкстры

Алгоритм Дейкстры решает задачу о кратчайших путях из одной вершины для взвешенного ориентированного графа G = (V, E) с исходной вершиной s, в котором веса всех ребер неотрицательны.

Алгоритм.

`d[u]` – расстояние до вершины u из исходной вершины s

`Q` – очередь с приоритетами. Извлекается вершина, для которой d[u] минимально

```
Initialize-Single-Source(G, s)
 for (для) всех вершин v из V[G]
     do d[v] = ∞
        родитель[v] = NIL
 d[s] = 0

Relax(u, v, w)
 if d[v] > d[u] + w(u, v)
     then d[v] = d[u] + w(u, v)
          предок[v] = u

Dijkstra(G, w, s)
 Initialize-Single-Source(G, s)
 S = Ø
 Q ← V[G]
 while Q <> Ø
     do u ← извлечь вершину, для которой d[u] минимально
        S = S U {u}
        for (для) всех вершин v смежных с u
            do Relax(u, v, w)
```

Суть алгоритма. d[u] – эта оценка пути по каждой из вершин на данном шаге. Изначально нам известно, что до исходной вершины путь 0, до остальных оценка на первом шаге бесконечность. На каждом шаге берем вершину u для которой оценка минимальна. Для неё можно доказать, что оценка является кратчайшим путем. Для смежных с ней вершин v, если оценка через вершину u оказывается меньше, чем текущая оценка вершины v, то уменьшаем оценку (релаксация).

Оценка времени работы. Оценка времени работы зависит от эффективности реализации очереди с приоритетами. При использовании массива, оценка получается $O(V^2)$; при использовании двоичной кучи оценка будет O(E log V); при использовании фибоначчиевой кучи оценка O(V log V + E).

Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда решает задачу о кратчайших весах из одной вершины для случая, когда весам ребер разрешено быть отрицательными. Этот алгоритм возвращает TRUE, если в графе нет цикла отрицательного веса, достижимого из исходной вершины, и FALSE, если таковой цикл имеется. В первом случае алгоритм находит кратчайшие пути и их веса; во втором случае кратчайших путей (по крайней мере, для некоторых вершин) не существует.

Алгоритм.

d[u] – расстояние до вершины u из исходной вершины s

```
Initialize-Single-Source(G, s)
 for (для) всех вершин v из V[G]
     do d[v] = ∞
        родитель[v] = NIL
 d[s] = 0

Relax(u, v, w)
 if d[v] > d[u] + w(u, v)
     then d[v] = d[u] + w(u, v)
          предок[v] = u
```

```
Bellman-Ford(G, w, s)
 Initialize-Single-Source(G, s)
 for i = 1 to |V[G]| - 1
     do for (для) каждого ребра (u, v) из E[G]
          do Relax(u, v, w)
 for (для) каждого ребра (u, v) из E[G]
     do if d[v] > d[u] + w(u, v)
          then return FALSE
 return TRUE
```

Оценка времени работы. Время работы алгоритма есть O(VE).

Алгоритм Флойда-Уоршолла

Может возникнуть задача, когда требуется найти кратчайшее расстояние между всеми парами вершин. Алгоритм Флойда-Уоршолла использует идею динамического программирования и позволяет за $O(V^3)$ найти кратчайшие пути между всеми парами вершин ориентированного взвешенного графа. Веса ребер могут быть отрицательными, но не допускается существования циклов отрицательного веса.

Алгоритм.

W – матрица весов n x n

D – матрица расстояний n x n

```
Floyd-Warshall(W)
D = W
for k = 1 to n
     do for i = 1 to n
          do for j = 1 to n
               do D[i, j] = min(D[i, j], D[i, k] + D[k, j])
```

Результатом работы алгоритма будет матрица D кратчайших расстояний между вершинами.

Остовные деревья

Связный подграф графа G, являющийся деревом и содержащий все его вершины, называют остовным деревом этого графа.

Минимальным остовным деревом является остовное дерево, сумма весов ребер которого минимальна.

Алгоритмы, предложенные ниже, будут основываться на следующем свойстве. Пусть G = (V, E) – связный граф с заданной функцией стоимости, определенной на множестве ребер. Обозначим через U подмножество множества вершин V. Если (u, v) – такое ребро наименьшей стоимости, что u Є U и v Є V \ U, тогда для графа G существует остовное дерево минимальной стоимости, содержащее ребро (u, v).

Алгоритм Прима

Q – очередь с приоритетами, ключом в которой является величина key[v] равная минимальному весу ребра из вершины v в вершину из множества уже обработанных

r – корень остовного дерева

```
MST-Prim(G, w, r)
 Q = V[G]
 for (для) каждой вершины u из Q
     do key[u] = ∞
 key[r] = 0
 предок[r] = NIL
```

```
while Q <> ∅
     do u ← извлечь вершину, для которой key[u] минимально
          for (для) каждой вершины v смежной с u
               do if (v Є Q) AND (w(u, v) < key[v])
                    then предок[v] = u
                         key[v] = w(u, v)
```

Оценка времени работы. Оценка времени работы зависит от эффективности реализации очереди с приоритетами. При использовании двоичной кучи оценка будет O(E log V); при использовании фибоначчиевой кучи оценка O(E + V log V).

Алгоритм Крускала

```
MST-Kruskal(G, w)
 A ← ∅
 for (для) каждой вершины v из V[G]
     do Make-Set(v)
 упорядочить рёбра E по весам
 for (для) (u, v) из E (в порядке возрастания веса)
     do if Find-Set(u) <> Find-Set(v)
          then A ← A U {(u, v)}
               Union(u, v)
 return A
```

Суть алгоритма. В начале каждый работы алгоритма каждая вершина графа лежит в своем множестве (имеет свой цвет). По ходу работы алгоритма просматриваются ребра в порядке возрастания весов. Если ребро соединяет вершины из разных множеств (разного цвета), то оно не создает цикла и, соответственно, может быть добавлено в дерево, которые мы строим.

Оценка времени работы. Оценка равна O(E log E), т.е. основное время уходит на сортировку. Предполагается, что для хранения непересекающихся множеств используется метод с объединением по рангу и сжатием путей.

Литература

1. Кормен Т. Алгоритмы: построение и анализ.
2. Ахо, Хопкрофт, Ульман. Структуры данных и алгоритмы.
3. Окулов С. Программирование в алгоритмах.