6 Хорошо известные графовые алгоритмы

6.1. Введение

Есть алгоритмы на графах, которые встречаются в различных приложениях настолько часто, что знать о них вы просто обязаны. Многие из них представляют собой различные способы сортировки графа или поиска подграфа с определенным свойством. Понимание относительных преимуществ этих фундаментальных алгоритмов — ключ к определению, когда использовать каждый из них в практических приложениях.

Эта глава начинается с описания алгоритмов поиска в ширину (Breadth-First Search, BFS) и в глубину (Depth-First Search, DFS), которые превращают произвольный граф в дерево поиска. В обоих случаях мы назначаем одну из вершин графа корнем дерева и рекурсивно исследуем ребра каждой вершины, пока все вершины графа[1] не будут добавлены в остовное

[1] То есть любая вершина, которой можно достичь, переходя по ребрам от корня; в случае связного графа это будут все вершины.

дерево[1]. Рассмотрим, для каких типов приложений полезен каждый вид дерева поиска, а затем перейдем к поиску кратчайших путей.

При анализе времени выполнения алгоритмов на графах обычно используются две системы обозначений. Одна из них — это обозначение числа вершин буквой n, а числа ребер — буквой m. Другая — указать размеры множеств вершин и ребер.

Для графа G множество вершин обозначается как $V(G)$ [читается «вершины G»), а множество ребер — как $E(G)$ (читается «ребра G»), поэтому $n = |V(G)|$, а $m = |E(G)|$.

Математическое предупреждение

Напомню, что в математике заключение значения в прямые скобки означает получение абсолютного значения или размера; здесь это означает размер множества, поэтому $|V(G)|$ можно прочитать как «размер множества вершин G». Если G подразумевается по умолчанию, то можно обращаться к множествам просто как V и E.

6.2. Поиск в ширину

При поиске в ширину мы сначала исследуем все вершины, которые примыкают к корню; затем исследуем

[1] Остовное дерево графа — это дерево, которое содержит все вершины графа.

все вершины, смежные с соседями корня, которые еще не были исследованы, и т. д. Таким образом, вначале мы исследуем все вершины, которые находятся на расстоянии k от корня (который обычно обозначается буквой s, от слова *source* — «источник»), и лишь затем переходим к вершинам на расстоянии $k + 1$.

Когда алгоритм завершает работу, глубина каждого узла дерева — это минимальное количество ребер (то есть длина кратчайшего пути), по которому можно добраться до этого узла из s как в дереве, так и в исходном графе. Чтобы найти этот путь, нужно идти по указателям на родительские узлы, пока не будет достигнут узел s.

Каково время выполнения поиска в ширину? При инициализации для каждой вершины инициализируются три упомянутых выше свойства, что занимает постоянное время для каждой вершины, в сумме $O(n)$. Затем каждая вершина добавляется в очередь, удаляется из очереди, и каждое из свойств изменяется не более одного раза — каждая из этих операций занимает $O(1)$ времени, так что для всех вершин снова получаем $O(n)$. Наконец, мы перебираем всех соседей каждой вершины, чтобы проверить, отмечены ли они, что занимает еще $O(m)$ времени (алгоритм 4). В сумме получаем общее время выполнения $O(n + m)$.

Для хранения графа требуется $n + m$ места в памяти, а очередь занимает $O(n)$ места, поэтому в сумме требуется пространство объемом $O(n + m)$. Это также означает, что алгоритм поиска в ширину работает за

линейное время по отношению к размеру входных данных (который равен $n + m$).

Алгоритм 4. Поиск в ширину

Входные данные: произвольный граф, корнем которого выбрана одна вершина, и все вершины имеют такие свойства:

- distance (расстояние) — изначально равно бесконечности;
- parent (родитель) — изначально равно нулю;
- marked (помечена) — изначально равно false.

Выходные данные: остовное дерево графа, каждая вершина которого максимально приближена к корню

```
begin
    Инициализировать очередь Q
    Задать s.distance = 0
    Пометить s
    Добавить в очередь s
    while Q не пустая do
        Удалить из очереди u
        foreach вершина v ∈ Adj(u) do
            if v.marked then
                продолжить
            end
            Задать v.parent = u
            Задать v.distance = u.distance + 1
            Пометить v
            Добавить в очередь v
        end
    end
end
```

6.3. Применение поиска в ширину

Как показано ранее, поиск в ширину позволяет найти длину кратчайшего пути от исходной вершины *s* до любой другой вершины графа за линейное время. Затем мы можем найти сам путь, следуя по родительским указателям от исходного узла до корня дерева, за время, пропорциональное длине пути. Это означает, что алгоритм поиска в ширину полезен для всех задач, где нужен поиск кратчайших путей. Вот несколько примеров таких задач.

6.3.1. Навигационные системы

Рассмотрим задачу построения маршрутов с помощью GPS. Если картографическая система хранит данные о локальной области в виде графа, вершинами которого являются адреса (или пересечения), а ребрами — улицы (или, точнее, короткие отрезки улиц), то она может выполнять поиск в ширину для дерева, источник которого — ваше текущее местоположение.

6.3.2. Проверка, является ли граф двудольным

При запуске поиска в ширину, если существует ребро между данной вершиной и уже отмеченной вершиной, расстояние которой либо такое же, либо меньше в степени 2, этим двум вершинам будет присвоен один и тот же цвет и граф не будет двудольным. Кроме того, ребро

вместе с одним или несколькими путями к наименьшему общему предку двух вершин является нечетным циклом, который будет признаком того, что граф не двудольный[1]. Двудольные графы используются в теории кодирования[2], в сетях Петри[3], в анализе социальных сетей и облачных вычислениях.

6.4. Поиск в глубину

Как и при поиске в ширину, при поиске в глубину мы начинаем с исходной вершины и рекурсивно проходим остальную часть графа (алгоритм 5). Но только теперь мы продвигаемся настолько глубоко, насколько можем, по одному пути и только потом исследуем другие пути.

Представьте, что вы изучаете собак. Студент, использующий алгоритм поиска в ширину, может начать с изучения названий всех пород — от австралийской короткохвостой пастушьей собаки до японского шпица — и лишь потом углубиться в детали. Тот же, кто использует поиск в глубину, начнет с австралийской короткохвостой, которая является пастушьей породой, поэтому он прочитает все о пастушьих собаках, что приведет его к чтению

---

[1] Сертификат — это доказательство того, что ответ, возвращаемый программой, является правильным; подробнее об этом вы прочитаете в главе 19.

[2] Теория кодирования изучает, в частности, криптографию, сжатие данных и исправление ошибок.

[3] Сети Петри используются для моделирования поведения систем.

о стадах, это, в свою очередь, — к истории о домашних животных, что приведет к... В общем, идею вы поняли. Если область поиска слишком велика или даже бесконечна, может использоваться модифицированная версия поиска в глубину, которая работает только до указанной глубины.

Алгоритм 5. Поиск в глубину

Входные данные: произвольный граф, у каждой вершины которого есть свойства distance (расстояние) (изначально равно нулю) и marked (помечена) (изначально имеет значение false), а также исходная вершина s

Выходные данные: остовное дерево графа

```
begin
    Инициализировать стек S
    S.Push(s)
    while S не пустой do
        u = S.Pop(s)
        if u.marked then
            продолжить
        end
        Пометить u
        foreach вершина v ∈ Adj(u) do
            if v.marked then
                продолжить
            end
            Задать v.parent = u
            S.Push(v)
        end
    end
end
```

Практическое применение

Недавно меня попросили помочь решить задачу планирования. Было несколько заданий, которые требовалось выполнить, со следующими ограничениями:

- ни одно задание нельзя начать раньше определенного времени;
- каждое задание имеет максимальное время выполнения;
- у задания может быть ноль или более заданий, от которых оно зависит, и ноль или более заданий, которые зависят от него; выполнение задания нельзя начать, пока не будут выполнены все задания, от которых оно зависит. Циклических зависимостей нет (поэтому это орграф частично упорядоченного множества);
- на выполнение всех заданий может уйти не более 24 часов.

Поиск кратчайшего пути — распространенная задача, но в данном случае мы фактически хотели найти самый длинный путь для каждой задачи, что и сделали, используя модификацию алгоритма поиска в глубину. Это позволило нам построить такой порядок выполнения заданий, который гарантировал, что во время выполнения любого задания все предки этого задания в графе зависимостей уже были выполнены, так что самое длительное время выполнения задания было равно сумме самого позднего времени, требуемого для выполнения любого из родителей данного задания, и количества времени, отведенного для самого задания. Повторное выполнение алгоритма в обратном порядке

позволило выявить все цепочки зависимостей, которые будут выполняться после выделенного времени, что позволило выяснить, какие задачи необходимо оптимизировать.

6.5. Кратчайшие пути

Рассмотрим задачу наискорейшего перемещения из одного места в другое. В теории графов это задача кратчайшего пути: найти маршрут между двумя вершинами, имеющий наименьший вес. В невзвешенном графе это просто путь с наименьшим количеством ребер; во взвешенном графе это путь с наименьшим суммарным весом ребер.

Рассмотрим вариации этой задачи.

- **Кратчайший путь из одной вершины.** Найти кратчайший путь от исходного узла ко всем остальным узлам графа. Пример: узнать кратчайший путь от пожарной части до каждой точки города.
- **Кратчайший путь в заданный пункт назначения.** Найти кратчайший путь от каждого узла графа до данной точки назначения. Это просто задача поиска кратчайшего пути из одной вершины, для которой направление всех ребер изменено на противоположное. Например, мы можем захотеть узнать кратчайший путь до больницы из любой точки города.
- **Кратчайший путь между всеми парами вершин.** Найти кратчайший путь между всеми парами узлов графа. В идеале наш GPS сможет построить наилучший маршрут из любой точки в любую точку.

6.5.1. Алгоритм Дейкстры

Алгоритм Дейкстры[1] изначально определял кратчайший путь между двумя узлами графа, но был расширен таким образом, что теперь он решает задачу поиска кратчайшего пути из одной вершины (алгоритм 6). Этот алгоритм применим к любому взвешенному графу (напомню, что невзвешенный граф — это просто взвешенный граф, у которого вес каждого ребра равен 1), у которого все ребра имеют неотрицательный вес.

Алгоритм 6. Алгоритм Дейкстры

Входные данные: граф G и исходная вершина s
Выходные данные: расстояние от s до любого другого узла графа G
Инвариантное: S — множество узлов, для которых определены кратчайшие пути

```
begin
    Инициализировать множество вершин S нулем
    Инициализировать очередь с приоритетами Q
    и поместить в нее все вершины G
    while Q не пустая do
        u = Q.ExtractMin()
        S = S ∪ {u}
        foreach вершина v ∈ Adj(u) do
            Relax(u, v, w)
        end
    end
end
```

[1] www-m3.ma.tum.de/twiki/pub/MN0506/WebHome/dijkstra.pdf.