Сетевое программирование

Лекция 15

Минимальное остовное дерево. Балансировка нагрузки в сети

Усіпов Н.М., PhD · КазНУ имени аль-Фараби

Цель и план лекции

Цель

Изучение понятия минимального остовного дерева и алгоритмов его построения, рассмотрение применения остовных деревьев в сетях и освоение основных подходов к балансировке нагрузки между каналами.

1. Введение
2. Минимальное остовное дерево
3. Алгоритмы построения
4. Применение остовных деревьев
5. Балансировка нагрузки
6. Варианты задачи об остовном дереве
7. Протоколы связующего дерева
8. Балансировка на разных уровнях

1 Введение

- Две задачи, рассматриваемые в данной лекции, отражают разные этапы жизни сети.

2 Минимальное остовное дерево

- Остовным деревом связного неориентированного графа называется подграф, который содержит все его вершины, является связным и не имеет циклов.
- Построение MST основано на двух свойствах:

2 Минимальное остовное дерево (продолжение)

Свойство разреза

Самое лёгкое ребро, пересекающее любой разрез графа, входит в минимальное остовное дерево (при различных весах).

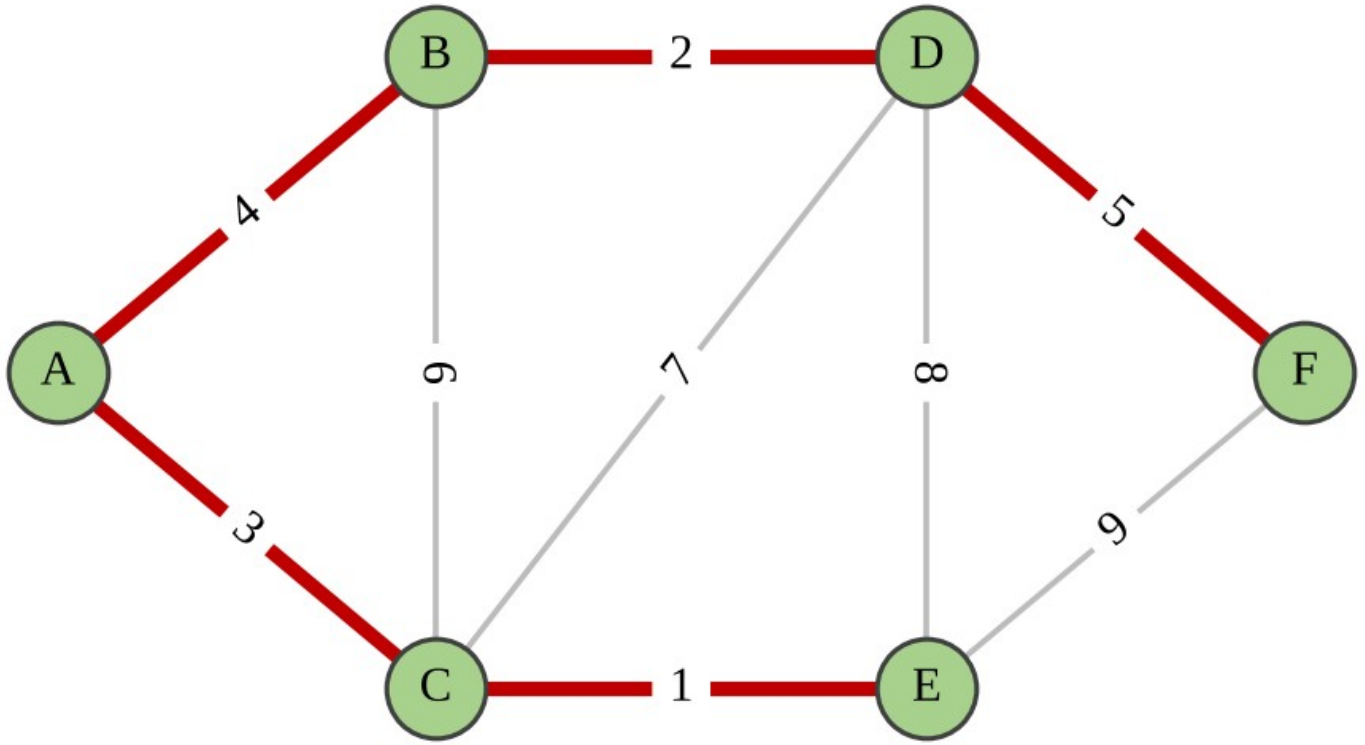
Свойство цикла

Самое тяжёлое ребро любого цикла не входит в минимальное остовное дерево (при различных весах).

2 Минимальное остовное дерево (продолжение)

- Если все веса различны, минимальное остовное дерево единственно. При наличии равных весов может существовать несколько деревьев одинаковой суммарной стоимости.

2 Минимальное остовное дерево (продолжение)



Минимальное остовное дерево (выделено красным)

3 Алгоритмы построения

Алгоритм Краскала (1956)

Рёбра сортируются по возрастанию веса и по очереди добавляются, если не образуют цикл.

Алгоритм Прима (1957)

Дерево выращивается от начальной вершины: на каждом шаге присоединяется самое лёгкое ребро, соединяющее дерево с остальной частью графа.

Алгоритм Борувки (1926)

Каждая компонента одновременно выбирает своё самое лёгкое исходящее ребро. Алгоритм удобен для параллельных вычислений.

В NetworkX применяется функция `minimum_spanning_tree`, а алгоритм выбирается параметром `algorithm`.

3 Построение минимального остовного дерева

Листинг 15.1.

```
import networkx as nx

G = nx.Graph()
G.add_weighted_edges_from([
    ("A", "B", 4), ("A", "C", 3), ("B", "C", 6), ("B", "D", 2),
    ("C", "D", 7), ("C", "E", 1), ("D", "E", 8), ("D", "F", 5),
    ("E", "F", 9)])

for alg in ("kruskal", "prim", "boruvka"):
    T = nx.minimum_spanning_tree(G, algorithm=alg)
    print(f"{alg:8} вес = {T.size(weight='weight'):g}")

T = nx.minimum_spanning_tree(G)
print("Рёбра:", sorted(tuple(sorted(e)) for e in T.edges))
print("Вес исходной сети:", G.size(weight="weight"))
print("Путь C-D в графе:", nx.dijkstra_path_length(G, "C", "D"))
print("Путь C-D в дереве:", nx.dijkstra_path_length(T, "C", "D"))
```

3 Результат: Построение минимального остовного дерева

```
kruskal  вес = 15
prim     вес = 15
boruvka  вес = 15
Рёбра: [('A', 'B'), ('A', 'C'), ('B', 'D'), ('C', 'E'), ('D', 'F')]
Вес исходной сети: 45.0
Путь C-D в графе: 7
Путь C-D в дереве: 9
```

При этом важно понимать, что минимальное остовное дерево минимизирует общую стоимость всего соединения, но не длину пути между конкретной парой узлов.

4 Применение остовных деревьев

Проектирование сетей

MST даёт нижнюю границу затрат на соединение всех узлов. Поскольку такая сеть не содержит резервных путей, каждое ребро в ней является мостом, поэтому на практике к дереву добавляют дополнительные каналы.

Протокол связующего дерева

Протокол STP, предложенный Радией Перлман, отключает избыточные соединения в коммутируемых сетях Ethernet, формируя логическое дерево.

Многоадресная рассылка

Дерево позволяет доставить данные множеству получателей без дублирования по одним и тем же каналам.

Кластеризация

Удаление самых тяжёлых рёбер из MST разделяет набор узлов на группы.

5 Балансировка нагрузки

- Балансировка нагрузки – это распределение трафика между несколькими путями или серверами для повышения эффективности использования ресурсов и предотвращения перегрузки.

$$\rho_e = f_e / c_e.$$

- Важным критерием служит максимальная загрузка каналов сети (MLU, Maximum Link Utilization), поскольку именно наиболее загруженный канал определяет появление очередей и потерь.

5 Балансировка нагрузки (продолжение)

ECMP

Трафик распределяется между всеми путями равной стоимости. Метод поддерживается протоколами маршрутизации и широко используется в центрах обработки данных.

Взвешенное распределение (WCMP)

Доля трафика на каждом пути выбирается с учётом пропускной способности, что полезно при неодинаковых каналах.

Распределение на уровне потоков

Чтобы пакеты одного соединения не приходили в неправильном порядке, маршрутизатор вычисляет хеш-функцию от полей заголовка и направляет весь поток по одному пути.

Управление трафиком (Traffic Engineering)

Маршруты явно назначаются с учётом требований и загрузки сети, например при использовании MPLS или программно-определяемых сетей (SDN).

5 Балансировка нагрузки (продолжение)

Балансировка серверов

Запросы распределяются между несколькими серверами с помощью алгоритмов циклического перебора, выбора наименее загруженного сервера и хеширования.

5 Сравнение одного пути и ECMP

Листинг 15.2.

```
G = nx.Graph()
for u, v in [("s", "a"), ("a", "t"), ("s", "b"), ("b", "t")]:
    G.add_edge(u, v, cap=10, cost=1)
G.add_edge("s", "c", cap=10, cost=3)
G.add_edge("c", "t", cap=10, cost=3)

DEMAND = 12

def utilization(paths):
    load = dict.fromkeys(map(frozenset, G.edges), 0.0)
    for p in paths:
        for u, v in zip(p, p[1:]):
            load[frozenset((u, v))] += DEMAND / len(paths)
    rho = {e: x / G.edges[tuple(e)]["cap"] for e, x in load.items()}
    return max(rho.values())

single = [nx.shortest_path(G, "s", "t", weight="cost")]
ecmp = list(nx.all_shortest_paths(G, "s", "t", weight="cost"))
print("Кратчайших путей:", len(ecmp))
print("MLU (один путь):", utilization(single))
print("MLU (ECMP):", utilization(ecmp))
```

6 Варианты задачи об остовном дереве

В практических постановках к задаче об остовном дереве добавляются дополнительные условия, которые могут существенно изменить её сложность.

Ограничение степени

Каждый узел может иметь не более заданного числа соединений, что соответствует количеству портов.

Дерево Штейнера

Необходимо соединить только заданное подмножество узлов, разрешая использование промежуточных.

Дерево кратчайших путей

Обеспечивает минимальное расстояние от корня до всех узлов, но не минимальную сумму весов.

Коммуникационное дерево

Минимизирует суммарный трафик с учётом потребностей пар узлов.

Минимальное и максимальное остовные деревья могут быть получены одним и тем же алгоритмом при обратном порядке сортировки.

6 Варианты задачи об остовном дереве (продолжение)

Сети с резервированием

Требуется получить минимальную по стоимости сеть, которая остаётся связной после отказа любого ребра.

Минимальное и максимальное остовные деревья могут быть получены одним и тем же алгоритмом при обратном порядке сортировки.

7 Протоколы связующего дерева

- Протокол STP работает в три этапа: выбирается корневой мост с наименьшим идентификатором, для каждого моста определяется корневой порт, обеспечивающий наименьшую стоимость до корня, а затем определяются назначенные порты сегментов.

8 Балансировка на разных уровнях

- Балансировка происходит на нескольких уровнях. В ядре сети применяется ECMP на основе хеширования пятёрки полей (адреса и порты отправителя и получателя, протокол).
- На уровне приложений балансировщик распределяет запросы между серверами. Алгоритмы циклического обхода и наименьшего числа соединений просты.
- Важно обеспечить устойчивость управления. Если маршрутизатор реагирует на загрузку слишком быстро, весь трафик может одновременно перейти на менее загруженный путь, который затем перегрузится.
- В программно-определяемых сетях контроллер получает данные о загрузке и на их основе изменяет распределение.

9 Разбор примера

1. Построим минимальное остовное дерево для рис. 15.1 алгоритмом Краскала.
2. Отсортированные рёбра: C–E (1), B–D (2), A–C (3), A–B (4), D–F (5), B–C (6), C–D (7), D–E (8), E–F (9).
3. Добавляем C–E, B–D, A–C и A–B, поскольку они не образуют циклов.
4. Добавляем D–F, и все шесть вершин оказываются соединены.
5. Суммарный вес равен 1 + 2 + 3 + 4 + 5 = 15.
6. Остальные рёбра образовали бы циклы.
7. Для WCMP при каналах 10 и 40 Гбит/с трафик 30 Гбит/с распределяется в отношении 1:4, то есть 6 и 24 Гбит/с.

9 Разбор примера (продолжение)

8 Загрузка обоих каналов равна 0,6.

9 При равном делении первый канал получил бы 15 Гбит/с и оказался бы перегружен.

10 Вопросы для обсуждения на занятии

Топология

Почему дерево не используется как окончательная топология магистральной сети?

Балансировка

Какие признаки свидетельствуют о плохой работе ECMP?

Устойчивость

Как избежать колебаний при адаптивной балансировке?

Контрольные вопросы

1. Дайте определение минимального остовного дерева.
2. Сформулируйте свойства разреза и цикла.
3. Опишите алгоритмы Краскала и Прима и сравните их сложность.
4. Почему минимальное остовное дерево не даёт кратчайших путей между узлами?
5. Для чего предназначен протокол связующего дерева (STP)?
6. Что такое максимальная загрузка канала и почему она важна?
7. В чём заключается метод ECMP? Почему распределение часто выполняется на уровне потоков?
8. Какими способами можно распределять нагрузку при неодинаковой пропускной способности каналов?

Список литературы

1. Kruskal J.B. On the shortest spanning subtree of a graph and the traveling salesman problem // Proceedings of the American Mathematical Society. – 1956. – Vol. 7, No. 1. – P. 48–50.
2. Prim R.C. Shortest connection networks and some generalizations // Bell System Technical Journal. – 1957. – Vol. 36, No. 6. – P. 1389–1401.
3. Borůvka O. O jistém problému minimálním // Práce Moravské přírodovědecké společnosti. – 1926. – Vol. 3. – S. 37–58.
4. Perlman R. An algorithm for distributed computation of a spanning tree in an extended LAN // ACM SIGCOMM Computer Communication Review. – 1985. – Vol. 15, No. 4. – P. 44–53.
5. Hopps C. Analysis of an Equal-Cost Multi-Path Algorithm: RFC 2992. – IETF, 2000.
6. Cormen T., Leiserson C., Rivest R., Stein C. Introduction to Algorithms. – 4th ed. – Cambridge, MA: MIT Press, 2022.
7. NetworkX Documentation [Электронный ресурс]. – URL: https://networkx.org/documentation/stable/