Сириус
IT-Колледж

Лекция 10. Транспортные сети

Максимальный поток. Алгоритм Форда – Фалкерсона.

Сириус IT-Колледж

Транспортная сеть

Сеть (flow network) $G = (V, E)$ представляет собой ориентированный граф, в котором каждое ребро $(u, v) \in E$ имеет положительную пропускную способность (capacity) $c(u, v) > 0$. Если $(u, v) \notin E$, предполагается что $c(u, v) = 0$.
В транспортной сети выделяются две вершины: исток s и сток t.

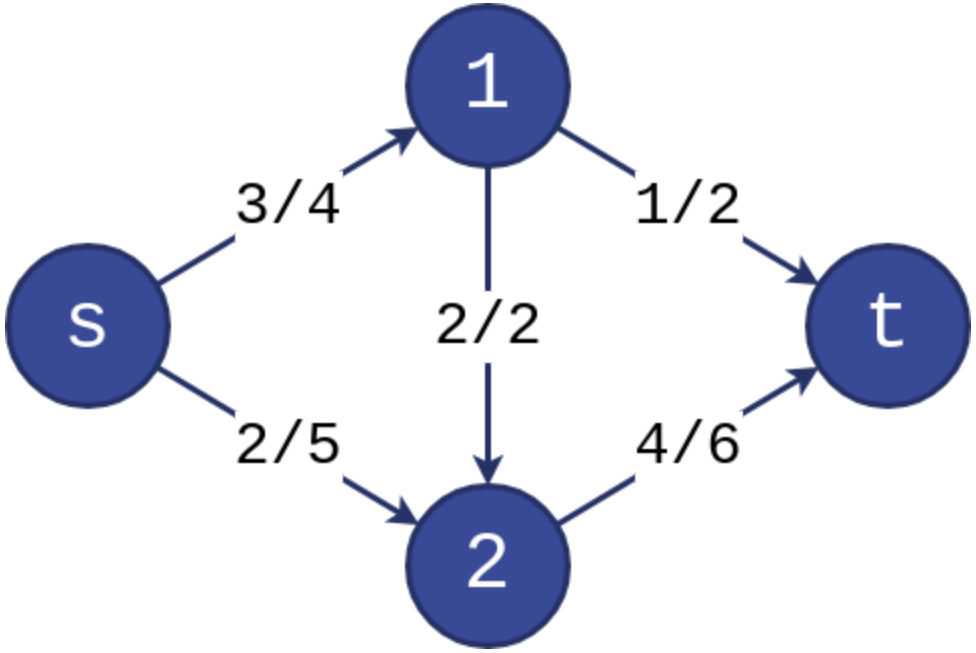
Потоком (flow) f в G является действительная функция $f : V \times V \to R$, удоволетворяющая условиям:

1. $f(u, v) = -f(v, u)$ (антисимметричность);
2. $f(u, v) \leqslant c(u, v)$ (ограничение пропускной способности), если ребра нет, то $f(u, v) = 0$;
3. $\sum_v f(u, v) = 0$ для всех вершин u, кроме s и t (закон сохранения потока).

Величина потока f определяется как $|f| = \sum_{v \in V} f(s, v)$.

Сириус
IT-Колледж

Транспортная сеть



Первое число означает величину потока, второе — пропускную способность ребра. Отрицательные величины потока не указаны (так как они мгновенно получаются из антисимметричности: $f(u, v) = -f(v, u)$). Сумма входящих рёбер везде (кроме источника и стока) равна сумме исходящих и на то, что в общем $c(u, v) \neq c(v, u)$. Кроме того, величина потока на ребре никогда не превышает пропускную способность этого ребра.

Величина потока в этом примере равна $3 + 2 = 5$ (считаем от вершины s).

Сириус
IT-Колледж

Максимальный поток

В теории оптимизации и теории графов, **задача о максимальном потоке** заключается в нахождении такого потока по транспортной сети, что сумма потоков из истока, или, что то же самое, сумма потоков в сток максимальна.

Сириус IT-Колледж

Алгоритм Форда-Фалкерсона.

1. Обнуляем все потоки. Остаточная сеть изначально совпадает с исходной сетью.
2. В остаточной сети находим любой путь из источника в сток. Если такого пути нет, останавливаемся.
3. Пускаем через найденный путь (он называется увеличивающим путём или увеличивающей цепью) максимально возможный поток:
    i. На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью $c_{\min}$.
    ii. Для каждого ребра на найденном пути увеличиваем поток на $c_{\min}$, а в противоположном ему — уменьшаем на $c_{\min}$.
    iii. Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных (антипараллельных) им рёбер, вычисляем новую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной сети, а если обнулилась, стираем его.
4. Возвращаемся на шаг 2.

Сириус
IT-Колледж

Важно, что алгоритм не конкретизирует, какой именно путь мы ищем на шаге 2 или как мы это делаем. По этой причине алгоритм гарантированно сходится только для целых пропускных способностей, но даже для них при больших значениях пропускных способностей он может работать очень долго. Если пропускные способности вещественны, алгоритм может работать бесконечно долго, не сходясь к оптимальному решению.

Если искать не любой путь, а кратчайший, получится алгоритм Эдмондса — Карпа или алгоритм Диница. Эти алгоритмы сходятся для любых вещественных весов за время $O(|V||E|^2)$ и $O(|V|^2|E|)$ соответственно.

Сириус
IT-Колледж

Алгоритм Эдмондса — Карпа

Алгоритм Эдмондса — Карпа — это вариант алгоритма Форда — Фалкерсона, при котором на каждом шаге выбирают кратчайший дополняющий путь из s в t в остаточной сети (полагая, что каждое ребро имеет единичную длину). Кратчайший путь находится поиском в ширину.

Сириус IT-Колледж

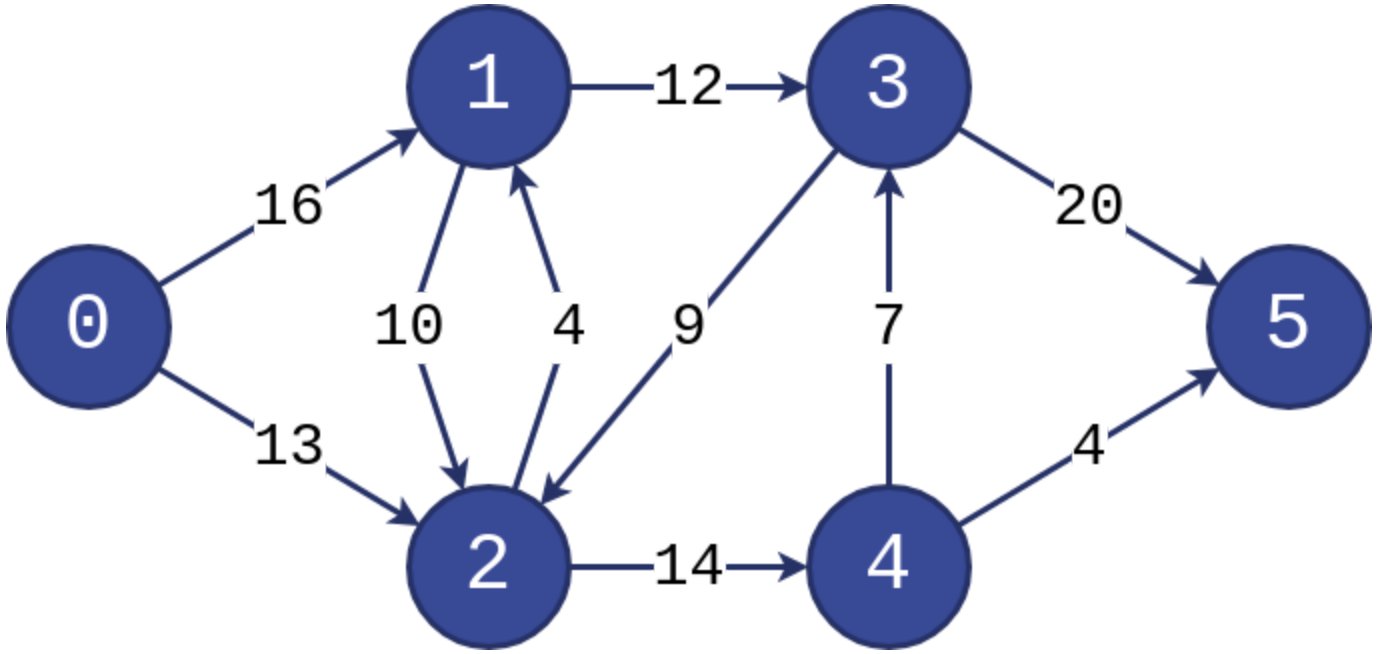
Алгоритм Эдмондса — Карпа

```
from collections import defaultdict

class Graph:
    def __init__(self, graph):
        self.graph = graph
        self. ROW = len(graph)

    def BFS(self, s, t, parent):
        visited = [False]*(self.ROW)
        queue = []

        queue.append(s)
        visited[s] = True

        while queue:u = queue.pop(0)
            for ind, val in enumerate(self.graph[u]):
                if visited[ind] == False and val > 0:
                    queue.append(ind)
                    visited[ind] = True
                    parent[ind] = u
                    if ind == t:
                        return True

        return False
```

```
    def FordFulkerson(self, source, sink):
        parent = [-1]*(self.ROW)
        max_flow = 0

        while self.BFS(source, sink, parent) :
            path_flow = float("Inf")
            s = sink
            while(s !=  source):
                path_flow = min (path_flow,
                    self.graph[parent[s]][s])
                s = parent[s]

            max_flow +=  path_flow

            v = sink
            while(v !=  source):
                u = parent[v]
                self.graph[u][v] -= path_flow
                self.graph[v][u] += path_flow
                v = parent[v]

        return max_flow
```

Сириус
IT-Колледж

Алгоритм Эдмондса — Карпа. Пример



Сириус
IT-Колледж

Алгоритм Эдмондса — Карпа. Пример