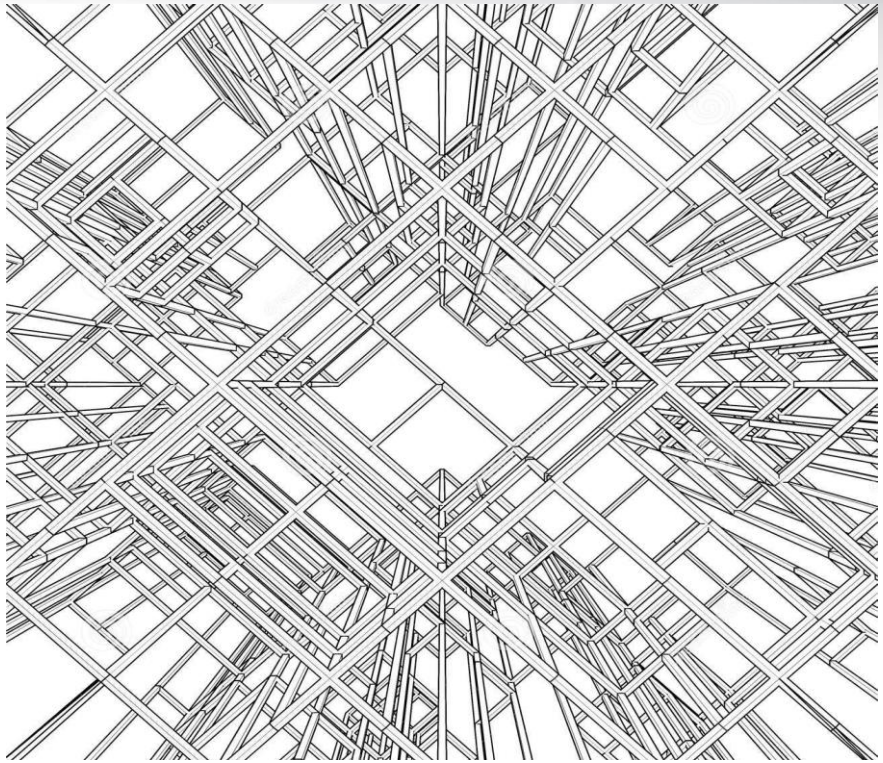
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Алгоритмы на графах ч.2



Беркунский Е.Ю., кафедра ИУСТ, НУК
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

Эйлеров цикл

В 1905 году был построен Императорский мост, который был впоследствии разрушен в ходе бомбардировки во время Второй мировой войны. Существует легенда о том, что этот мост был построен по приказу самого кайзера, который не смог решить задачу мостов Кёнигсберга и стал жертвой шутки, которую сыграли с ним учёные умы, присутствовавшие на светском приёме (если добавить восьмой мост, то задача становится разрешимой). На опорах Императорского моста в 2005 году был построен Юбилейный мост.

Эйлеров цикл в графе

- **Эйлеров путь** (эйлерова цепь) в графе — это путь, проходящий по всем рёбрам графа и притом только по одному разу.
- **Эйлеров цикл** — это эйлеров путь, являющийся циклом.
- **Эйлеров граф** — граф, содержащий эйлеров цикл.
- **Полуэйлеров граф** — граф, содержащий эйлеров путь (цепь).

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Поиск эйлерова цикла в графе

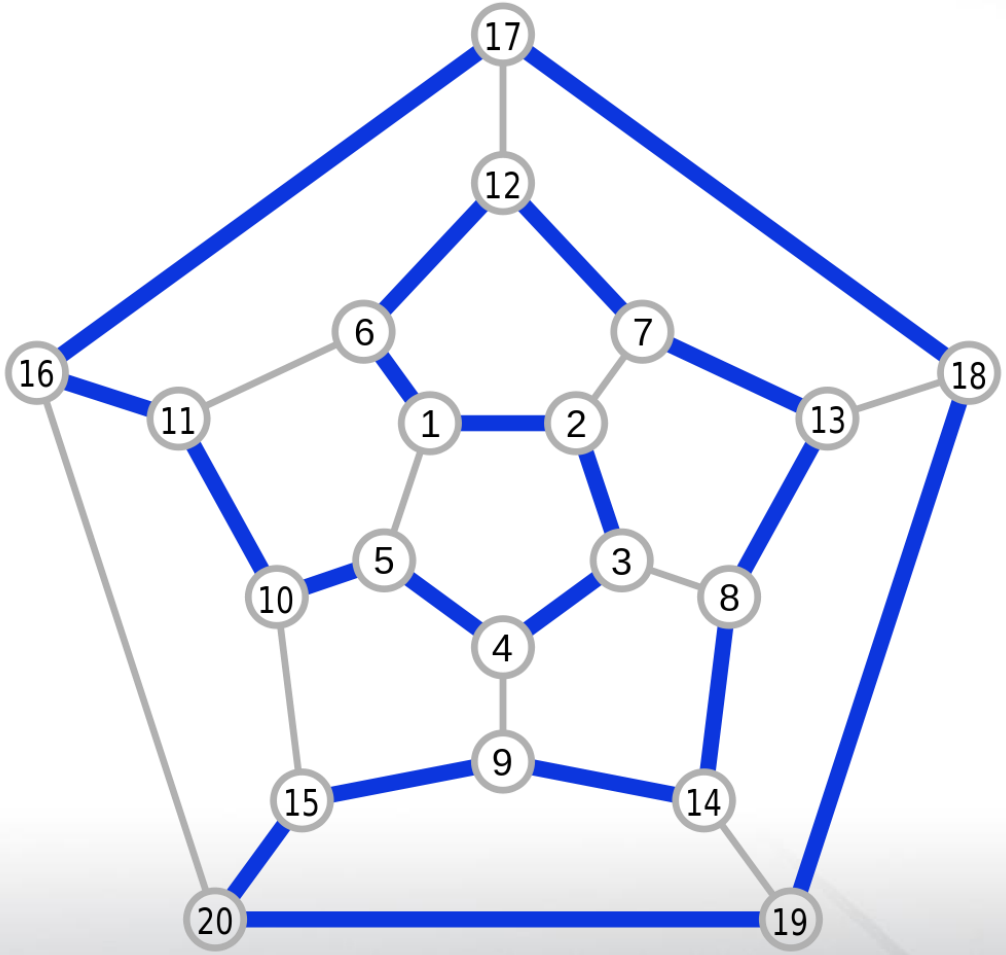
```
procedure find_all_cycles (v)
var массив cycles
```

1. пока есть цикл, проходящий через v, находим его добавляем все вершины найденного цикла в массив cycles (сохраняя порядок обхода) удаляем цикл из графа
2. идем по элементам массива cycles каждый элемент cycles[i] добавляем к ответу из каждого элемента рекурсивно вызываем себя: find_all_cycles (cycles[i])

Гамильтонов цикл

- **Гамильтонов граф** — это граф, содержащий *гамильтонову цепь* или *гамильтонов цикл*.
- **Гамильтонов путь** (или **гамильтонова цепь**) — путь (цепь), содержащий каждую вершину графа ровно один раз. Гамильтонов путь, начальная и конечная вершины которого совпадают, называется **гамильтоновым циклом**. Гамильтонов цикл является простым остовным циклом
- Гамильтоновы путь, цикл и граф названы в честь ирландского математика У. Гамильтона, который впервые определил эти классы, исследовав задачу «кругосветного путешествия» по додекаэдру, узловые вершины которого символизировали крупнейшие города Земли, а рёбра — соединяющие их дороги.

Гамильтонов цикл



Гамильтонов цикл

Эвристические оптимизации

- При прямом переборе вариантов возможно значительное увеличение средней сложности поиска гамильтонова пути на случайных графах
- Для улучшения данного способа можно на каждом шаге перебора при некоторой построенной части цепи проверять, образуют ли оставшиеся вершины связный граф (если не образуют, то цепь не может являться началом гамильтоновой цепи); на каждом шаге перебора при выборе следующей вершины пробовать сначала вершины с наименьшей остаточной степенью (количеством рёбер, ведущих в ещё не посещённые вершины).
- Кроме того, если это дерево является цепью, то гамильтонов цикл в нём возможен. Иначе (если в дереве есть вершины степенью не меньше, чем 3) построение гамильтонова цикла невозможно

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Экстремальные задачи теории графов

Задача коммивояжёра

- Коммивояжёру необходимо посетить каждый город в пределах некоторой территории и возвратиться в пункт отправления. Требуется, чтобы путь был как можно короче. Таким образом, исходная задача преобразуется в задачу поиска минимальной протяженности (длительности или стоимости)[1].
- Задачу можно переформировать в терминах теории графов — построить такой граф G(X, A), вершины которого соответствуют городам, а ребра — коммуникации между городами. Решение этой задачи ищут среди гамильтоновых циклов построенного графа.
- Известно много способов решения этой задачи. Можно выделить методы, разработанные Беллмором и Немхаузером[2], Гарфинкелем и Немхаузером[3], Хелдом и Карпом[4], Стекханом[5]. Также известными решениями задачи коммивояжёра являются метод ветвей и границ и метод последовательного улучшения решения

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Построение минимального остовного дерева

- Алгоритм Крускала
- Алгоритм Прима
- Алгоритм Борувки

Алгоритм Крускала

- Вначале текущее множество рёбер устанавливается пустым.
- Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству.
- Когда таких рёбер больше нет, алгоритм завершён.

Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса.

Алгоритм Прима

- Построение начинается с дерева, включающего в себя одну (произвольную) вершину.
- В течение работы алгоритма дерево разрастается, пока не охватит все вершины исходного графа.
- На каждом шаге алгоритма к текущему дереву присоединяется самое лёгкое из рёбер, соединяющих вершину из построенного дерева, и вершину не из дерева.

Алгоритм Борувки

- Изначально, пусть *T* — пустое множество ребер (представляющее собой остовный лес, в который каждая вершина входит в качестве отдельного дерева).
- Пока *T* не является деревом (что эквивалентно условию: пока число рёбер в *T* меньше, чем *V* − 1, где *V* — число вершин в графе):
  - Для каждой компоненты связности (то есть, дерева в остовном лесе) в подграфе с рёбрами *T*, найдём самое дешёвое ребро, связывающее эту компоненту с некоторой *другой* компонентой связности. (Предполагается, что веса рёбер различны, или как-то дополнительно упорядочены так, чтобы всегда можно было найти единственное ребро с минимальным весом).
  - Добавим все найденные рёбра в множество *T*.
- Полученное множество рёбер *T* является минимальным остовным деревом входного графа.

Задача о максимальном потоке

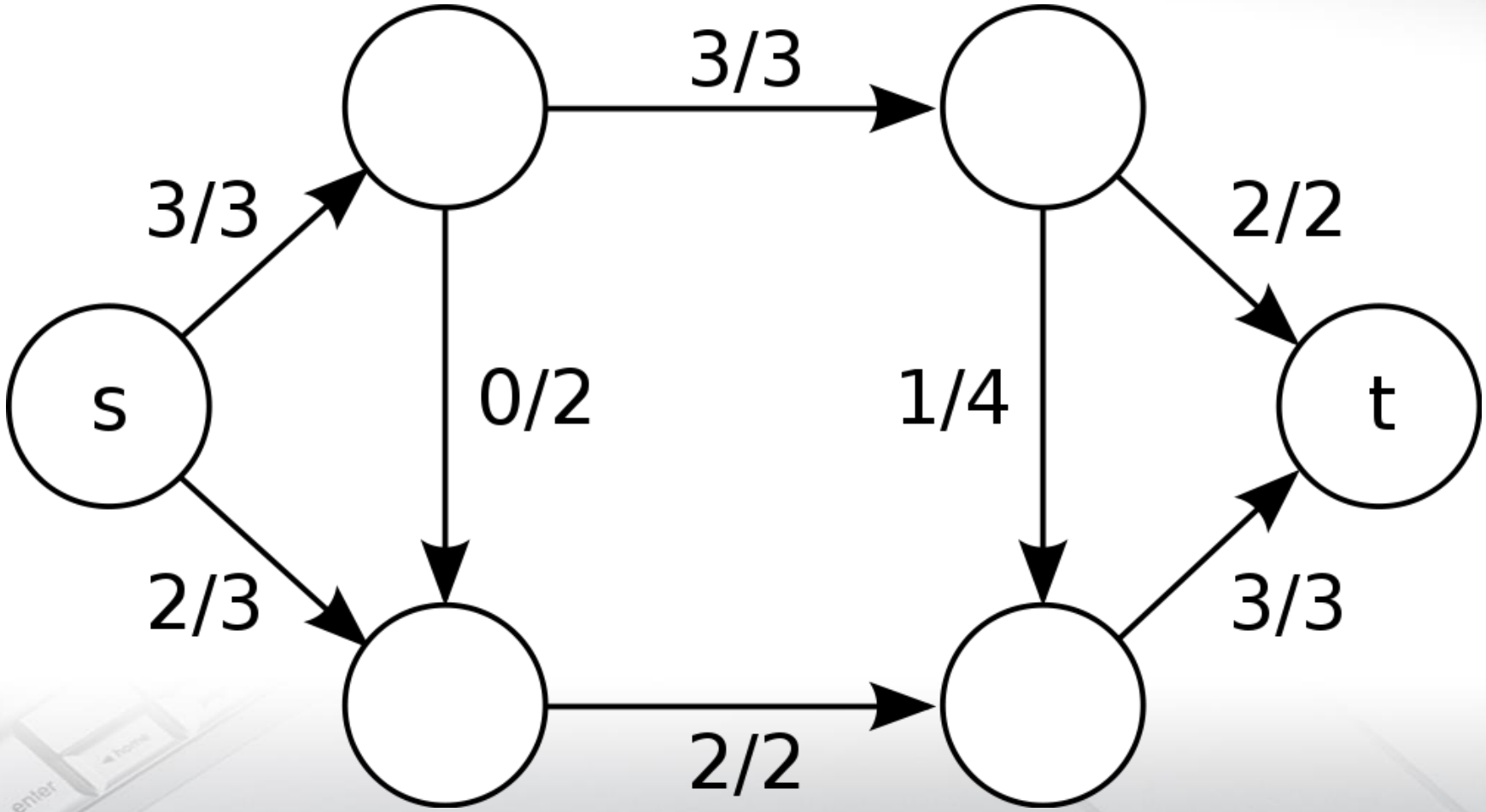
Дана транспортная сеть N = (V,E) с источником s ϵ V, стоком t ϵ V и пропускными способностями *c*.

Величиной потока (**value of flow**) называется сумма потоков из источника $|f| = \sum_{v \epsilon V} f_{sv}$.

В статье «Транспортная сеть» доказано, что она равна сумме потоков в сток $\sum_{w \in V} f(w, t)$.

Задача о максимальном потоке заключается в нахождении такого потока, что его величина максимальна.

Задача о максимальном потоке



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Решения

- Линейное программирование
- Алгоритм Форда-Фалкерсона
- Алгоритм Эдмондса-Карпа
- Алгоритм Диница
- Алгоритм проталкивания предпотока
- Алгоритм «поднять в начало»

Задача о максимальном потоке

Решение этой задачи с использованием методов Линейного программирования имеет экспоненциальную сложность, а поэтому, не представляет интереса.

Задача о максимальном потоке

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:
   - На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью c_{min}.
   - Для каждого ребра на найденном пути увеличиваем поток на c_{min}, а в противоположном ему - уменьшаем на c_{min}.
   - Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных 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.

Если искать не любой путь, а кратчайший, получится алгоритм Эдмондса-Карпа или алгоритм Диница.

Задача о максимальном потоке

Дан граф $G(V,E)$ с пропускной способностью $c(u,v)$ и потоком $f(u,v) = 0$ для ребер из u в v. Необходимо найти максимальный поток из источника s в сток t. На каждом шаге алгоритма действуют те же условия, что и для всех потоков:

- $f(u,v) \le c(u,v)$. Поток из u в v не превосходит пропускной способности.
- $f(u,v) = -f(v,u)$.
- $\sum_v f(u,v) = 0 \iff f_{in}(u) = f_{out}(u)$ для всех узлов u, кроме s и t.

Поток не изменяется при прохождении через узел.

- **Остаточная сеть** $G_f(V,E_f)$ — сеть с пропускной способностью $c_f(u,v) = c(u,v) - f(u,v)$ и без потока.
- **Вход** Граф с пропускной способностью c, источник s и сток t
  Выход Максимальный поток f из s в t

- $f(u,v) \leftarrow 0$ для всех ребер (u,v)
- Пока есть путь p из s в t в G_f, такой что $c_f(u,v) > 0$ для всех ребер (u,v)ϵ p:
  - Найти $c_f(p) = \min\{c_f(u,v) | (u,v) \in p\}$
  - Для каждого ребра $(u,v) \epsilon p$

  $f(u,v) \leftarrow f(u,v) + c_f(p)$ и $f(v,u) \leftarrow f(v,u) - c_f(p)$
- Путь может быть найден, например, поиском в ширину

Задача о максимальном потоке

Алгоритм Диница

Пусть G=((V,E),c,s,t)} — транспортная сеть, в которой:
c(u,v) - пропускная способность и
f(u,v) - поток через ребро (u,v).

Остаточная пропускная способность — отображение $c_f: V \times V \to R^+$

1. Если $(u, v) \in E$,

$$c_f(u, v) = c(u, v) - f(u, v)$$
$$c_f(v, u) = f(u, v)$$

2. $c_f(u, v) = 0$ иначе.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Задача о максимальном потоке

Алгоритм Диница

Остаточная сеть — граф $G_f = ((V, E_f), c_f|_{E_f}, s, t)$, где

$$E_f = \{(u, v) \in V \times V \mid c_f(u, v) > 0\}.$$

Дополняющий путь — $s - t$ путь в остаточном графе G_f.

Пусть $dist(v)$ — длина кратчайшего пути из s в v в графе G_f.
Тогда вспомогательная сеть графа G_f — граф $G_L = (V, E_L, c_f|_{E_L}, s, t)$
где

$$E_L = \{(u, v) \in E_f \mid \mathrm{dist}(v) = \mathrm{dist}(u) + 1\}$$

Блокирующий поток — s-t поток f такой, что граф

$G' = (V, E'_L, s, t)$ с $E'_L = \{(u, v) \mid f(u, v) < c_f|_{E_L}(u, v)\}$ не содержит s-t пути

Задача о максимальном потоке

Алгоритм Диница

Вход: Сеть $G = ((V, E), c, s, t)$.

Выход: $s - t$ поток f максимальной величины.

1. Установить $f(e) = 0$ для каждого $e \in E$.
2. Создать G_L из G_f графа G. Если $\mathrm{dist}(t) = \infty$, остановиться и вывести f.
3. Найти блокирующий поток f' в G_L.
4. Дополнить поток f потоком f' и перейти к шагу 2.

Задача о максимальном потоке

Алгоритм Диница

- Каждый раз количество рёбер в блокирующем потоке увеличивается хотя бы на одно, поэтому в алгоритме не более n-1 блокирующих потоков, где n — количество вершин в сети.
- Вспомогательная сеть G_L может быть построена обходом в ширину за время $O(E)$, а блокирующий поток на каждом уровне графа может быть найден за время $O(VE)$.
- Поэтому время работы алгоритма Диница есть $O(V)*(O(E)+O(VE))=O(V^2E)$.

Используя динамические деревья, можно находить блокирующий поток на каждой фазе за время $O(E \log V)$, тогда время работы алгоритма Диница может быть улучшено до $O(VE \log V)$