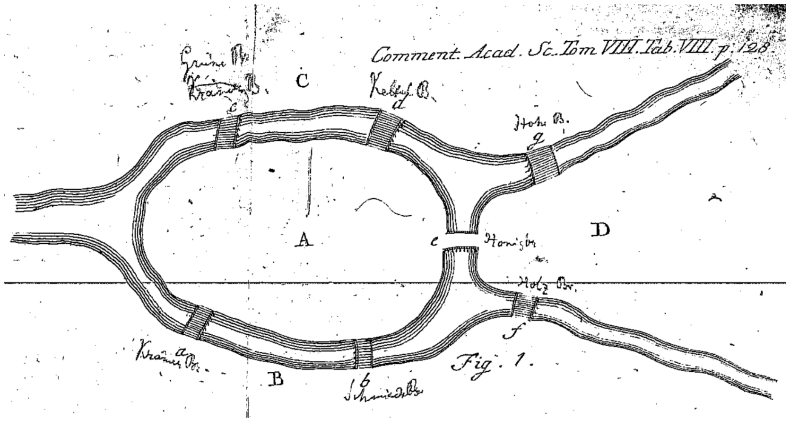
Адрианов Н.М.
Иванов А.Б.

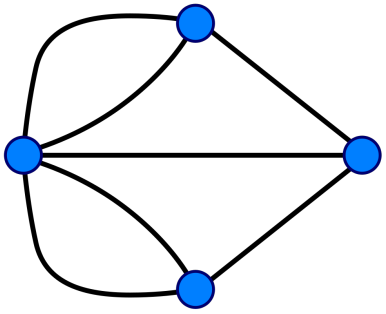
Алгоритмы и структуры данных

Графы. DFS и BFS. Топологическая сортировка. Сильносвязанные компоненты.

Задача о Кёнигсбергских мостах

Эйлер (1736)





Основные определения

Граф (V, E) V – множество вершин
E – множество ребер

Ориентированный: $E = \{(u, v) \mid u, v \in V\}$
Неориентированный: $E = \{(u, v) \mid u, v \in V\} / \sim \qquad (u, v) \sim (v, u)$

Степень вершины
Ориентированный: $d_{out}(v)$, $d_{in}(v)$

$$\sum_{v \in V} d_{out}(v) = \sum_{v \in V} d_{in}(v) = |E|$$

Неориентированный: $d(v)$

$$\sum_{v \in V} d(v) = 2|E|$$

Приложения: графы везде

- Веб-граф
- Графы социальных сетей
- Компьютерные и электросети
- Дорожная сеть
- Управление проектами
- ...

Пути и связность

Путь от вершины u к вершине v: $e_1, e_2, \dots e_k \in E$

$$e_i = (u_{i-1}, u_i), \quad u_0 = u, \quad u_k = v$$

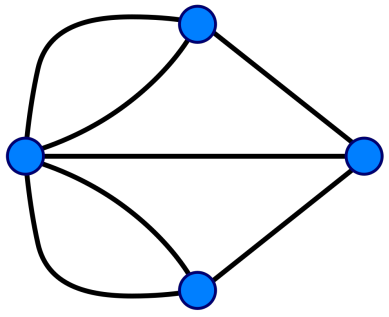
Цикл: путь, в котором начало и конец совпадает

Связный граф: для любых $u, v \in V$ $\exists$ путь от u к v

Компонента связности: максимальное $C \subset V$ такое, что для любых $u, v \in C$ $\exists$ путь от u к v

Дерево: связный неориентированный граф без циклов

Задача о Кёнигсбергских мостах



Эйлеров путь: путь без повторяющихся ребер

Существует ли эйлеров путь?

Эйлеров путь существует $\Leftrightarrow$
в графе не более 2 вершин нечетной степени

Гамильтонов путь: путь без повторяющихся вершин

Существует ли гамильтонов путь?

NP-полная задача

Когда граф можно уложить на плоскость?

Теорема Куратовского, 1930:
Граф является планарным тогда и только тогда, когда в нем нет топологических миноров, изоморфных K_5 или $K_{3,3}$.

Граф H является топологическим минором G если существует гомеоморфное вложение H в G, т.е. существует подграф H' в G такой, что H' получается из H подразделением ребер.

Алгоритм: гамма-алгоритм

Карты на поверхностях

X – поверхность рода g (сфера с g ручками).

Карта – граф G, вложенный в X так, что дополнение $X \setminus G$ распадается на диски (страны / грани).

Формула Эйлера (V – вершины, E – ребра, F – страны):

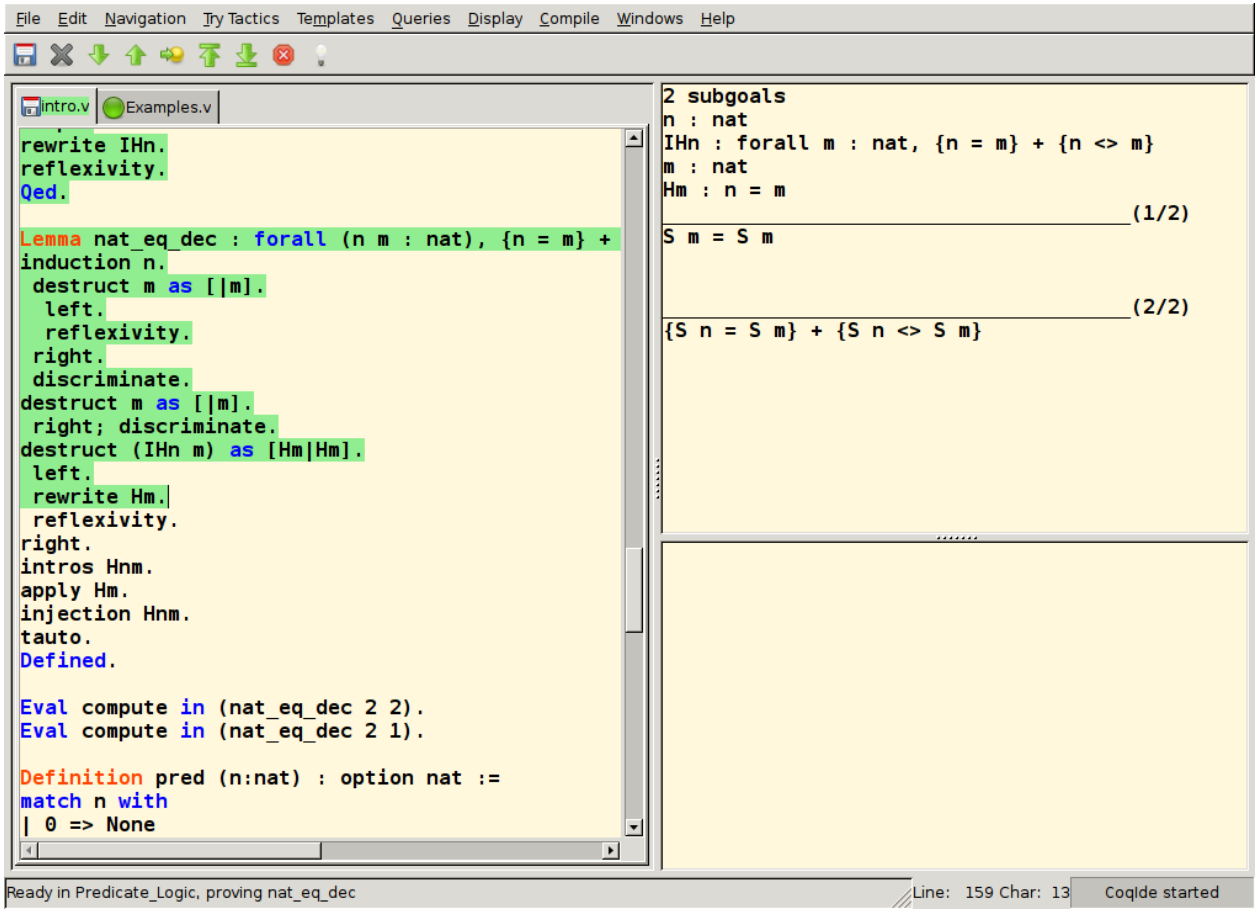
$$2 - 2g = V - E + F$$

Проблема 4 красок

Сколько красок необходимо, чтобы раскрасить плоскую карту так, чтобы граничащие страны имели разный цвет?

- 5 красок достаточно: Heawood, 1890
- 4 краски: Appel, Haken, 1976
  - любой контрпример содержит минимальный
  - 1936 карт, которые могли бы быть минимальными контрпримерами
  - сотни страниц доказательства + программа
- более простое доказательство (также с использованием компьютера): Robertson, Sanders, Seymour, Thomas, 1997
- формализованное доказательство в системе Coq: Gonthier, 2005

Система Coq: interactive theorem prover



Gonthier + team, 2012: теорема Фейта-Томпсона (1962)

Не существует простых неабелевых групп нечетного порядка

Перечислительные задачи

Сколько существует плоских деревьев с $n+1$ ребрами и выделенным корнем?

Числа Каталана

$$c_{n+1} = c_0 c_n + c_1 c_{n-1} + \ldots + c_{n-1} c_1 + c_n c_0, \quad c_0 = 1$$

$$F(z) = c_0 + c_1 z + c_2 z^2 + \ldots$$

$$F^2(z) = c_0 c_0 + (c_0 c_1 + c_1 c_0) z + (c_0 c_2 + c_1 c_1 + c_2 c_0) z^2 + \ldots$$

$$F(z) = 1 + z F^2(z)$$

$$F(z) = \frac{1 \pm \sqrt{1 - 4z}}{2z}$$

$$c_n = \frac{(2n)!}{n!(n+1)!} = \frac{1}{n+1} C_{2n}^n$$

Теория Рамсея

На вечеринке встретились 6 человек. Докажите, что среди них найдутся трое попарно знакомых или трое попарно незнакомых.

Теория Рамсея

Для любых $n, m \in \mathbb{N}$ существует $R(m, n)$ такое, что для любого $N \geq R(m, n)$ в полном графе K_N, чьи ребра раскрашены в красный и синий цвета, найдется либо полный подграф K_n красного цвета, либо полный подграф K_m синего цвета.

	3	4	5	6	7	8	9
3	6						
4	9	18					
5	14	25	43-49				
6	18	36-41	58-87	102-165			
7	23	49-61	80-143	113-298	205-540		
8	28	58-84	101-216	132-495	217-1031	282-1870	
9	36	73-115	126-316	169-780	241-1713	317-3583	565-6588

Теорема Ван дер Вардена

Для любого $n \in \mathbb{N}$ существует N такое, что для любой раскраски чисел $1, 2, \ldots N$ в два цвета из этих чисел можно выбрать арифметическую прогрессию длины n одного цвета.

$$
\begin{aligned}
\mathrm{DOUBLE}(n) &= 2 + 2 + \ldots + 2 = 2n \\
\mathrm{EXP}(n) &= \mathrm{DOUBLE}(\mathrm{DOUBLE}(\ldots \mathrm{DOUBLE}(1) \ldots)) = 2^n \\
\mathrm{TOWER}(n) &= \mathrm{EXP}(\mathrm{EXP}(\ldots \mathrm{EXP}(1) \ldots)) = 2^{2^{2^{\cdots^2}}} \\
\mathrm{WOW}(n) &= \mathrm{TOWER}(\mathrm{TOWER}(\ldots \mathrm{TOWER}(1) \ldots))
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{ACKERMAN}(1) &= \mathrm{DOUBLE}(1) \\
\mathrm{ACKERMAN}(2) &= \mathrm{EXP}(2) \\
\mathrm{ACKERMAN}(3) &= \mathrm{TOWER}(3) \\
\mathrm{ACKERMAN}(4) &= \mathrm{WOW}(4) \\
&\ldots
\end{aligned}
$$

Представление графа в программе

Матрица смежности

Списки смежности

Поиск в глубину (DFS = Depth First Search)

```
procedure DFS( G, s, t ):
  if s = t:
    return true
  visited = array of size G.V
  return DFS( G, s, t, visited )
end

procedure DFS( G, s, t, visited ):
  visited[s] = true
  for x in G.neighbours(s):
    if x = t:
      return true
    if not visited[x]:
      if DFS(G, x, t, visited):
        return true
  end
  return false
end
```

Поиск в ширину (BFS)

```
procedure BFS( G, s, t ):
  if s = t:
    return true

  visited = array of size G.V
  Q = queue

  visited[s] = true
  Q.push(s)

  while not Q.isEmpty:
    y = Q.pop()
    for x in G.neighbours(y):
      if x = t:
        return true
      if not visited[x]:
        visited[x] = true
        Q.push(x)
      end
    end
  end
  return false
end
```

DFS без рекурсии

Заменить в алгоритме BFS очередь на стек!

Полный обход всего графа (DFS)

Утв. После вызова `DFS(G, x)` для всех вершин `y`, достижимых из `x`, `visited[y] = true`

```
procedure DFS( G ):
  visited = array bool[G.V]
  for x in G.vertices:
    if not visited[x]:
      DFS(G, x)
  end
end

procedure DFS( G, x ):
  visited[x] = true
  Previsit(x)
  for y in G.neighbours(x):
    if not visited[y]:
      DFS(G, y)
  end
  Postvisit(x)
end
```

Компоненты связности неориент.графа

```
procedure Previsit( x ):
  ccnum[x] = cc
end
```

```
procedure DFS( G ):
  cc = 0
  ccnum = array int[G.V]
  visited = array bool[G.V]
  for x in G.vertices:
    if not visited[x]:
      DFS(G, x)
      cc = cc + 1
  end
end

procedure DFS( G, x ):
  visited[x] = true
  Previsit(x)
  for y in G.neighbours(x):
    if not visited[y]:
      DFS(G, y)
  end
  Postvisit(x)
end
```

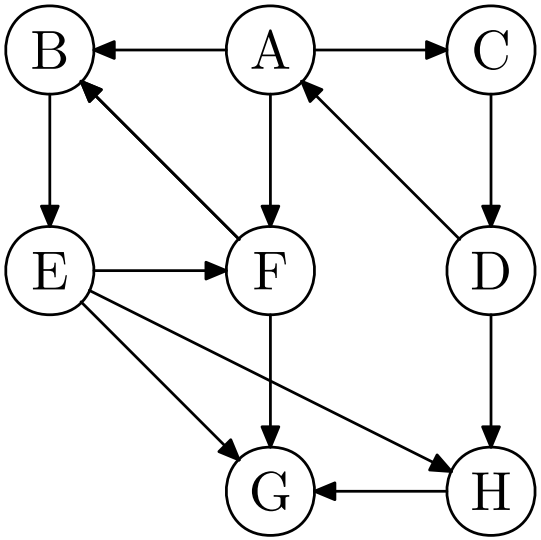
DFS в ориентированных графах

```
procedure Previsit( x ):
  pre[x] = clock
  clock = clock + 1
end

procedure Postvisit( x ):
  post[x] = clock
  clock = clock + 1
end
```

```
procedure DFS( G ):
  pre = array int[G.V]
  post = array int[G.V]
  clock = 1
  ...
end
```

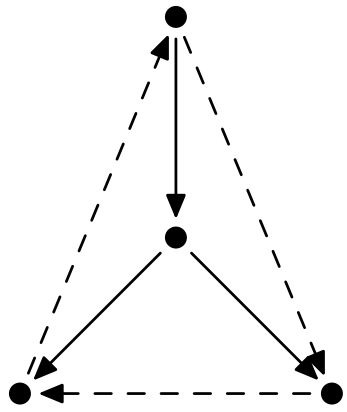
DFS в ориентированных графах




```
A: B,C,F        E: F,G,H
B: E            F: B,G
C: D            G: -
D: A,H          H: G
```

Типы ребер

древесное/прямое	`[`	`[`	`]`	`]`
	`u`	`v`	`v`	`u`
обратное	`[`	`[`	`]`	`]`
	`v`	`u`	`u`	`v`
перекрестное	`[`	`]`	`[`	`]`
	`v`	`v`	`u`	`u`



Утв 1. Для любых вершин `u` и `v` отрезки `[pre(u), post(u)]` и `[pre(v), post(v)]` либо вложены, либо не пересекаются.

Утв 2. Ориентированный граф содержит цикл $\Leftrightarrow$ существует обратное ребро.

Утв 3. Для любого ребра `(u,v)` в ориентированном ациклическом графе (DAG) `post(u) > post(v)`.

Топологическая сортировка

Дано: Ориентированный ациклический граф $G = (V, E)$.

Надо: Пронумеровать вершины V графа так, чтобы для любого ребра $(u, v) \in E$ номер вершины v был больше номера вершины u.

(Другими словами: расположить вершины на временной оси так, чтобы все ребра были направлены “в будущее”.)

Топологическая сортировка

Дано: Ориентированный ациклический граф $G = (V, E)$.

Надо: Пронумеровать вершины V графа так, чтобы для любого ребра $(u, v) \in E$ номер вершины v был больше номера вершины u.

(Другими словами: расположить вершины на временной оси так, чтобы все ребра были направлены "в будущее".)

Решение: Выводим вершины в обратном post-порядке.

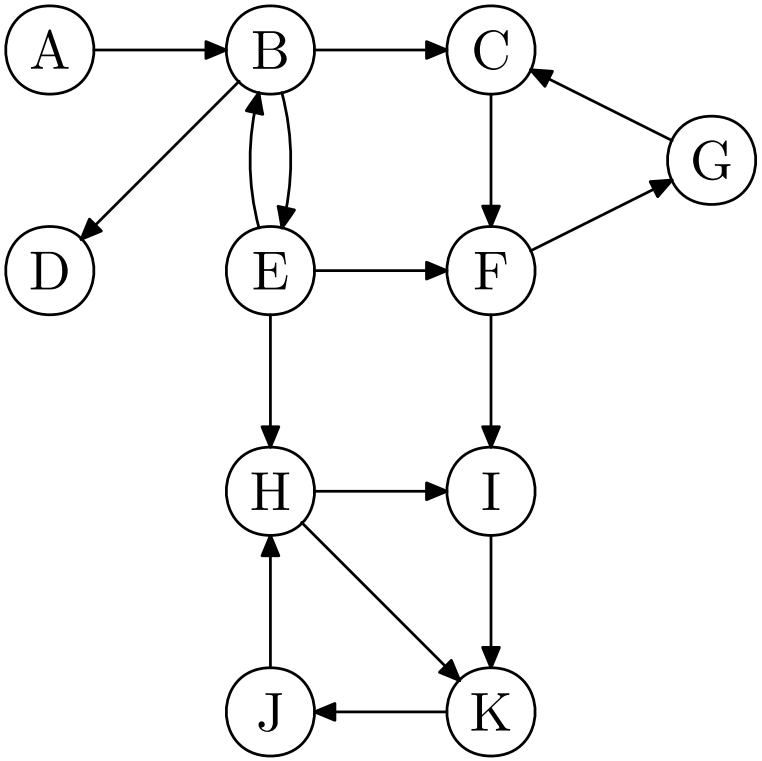
Истоки и стоки в DAG

Вершина называется *истоком*, если не существует входящих в нее ребер.

Вершина называется *стоком*, если не существует исходящих из нее ребер.

Утв. В ориентированном ациклическом графе (DAG) существует хотя бы один исток и хотя бы один сток.

Компоненты сильной связности



Компоненты сильной связности

Утв 1. Пусть `v` лежит в компоненте-стоке. Тогда `DFS(G, v)` обойдет все вершины в компоненте и только их.

Компоненты сильной связности

Утв 2. Пусть C и C' – компоненты сильной связности, и существует ребро из C в C'. Тогда

$$\max_{c \in C} \texttt{post}(c) > \max_{c' \in C'} \texttt{post}(c')$$

Следствие. Вершина с максимальным post-значением лежит в компоненте-истоке.

Алгоритм Косарайю

(S. Rao Kosaraju, Micha Sharir, 1979)

G^R – обращенный граф

1. Запустить DFS для G^R.
2. Запустить алгоритм нахождения компонент G (как в неориентированном случае) в обратном post-порядке.

Robert Tarjan – алгоритм с однократным DFS, 1972

2-выполнимость (2-SAT)

Дано: булева формула в КНФ такая, что в каждом терме только 2 литерала:

$$F(x_1, x_2, \ldots x_n) = \bigwedge_{\alpha} (z_\alpha \vee w_\alpha), \qquad \text{где } z_\alpha, w_\alpha = x_i \text{ или } \bar{x}_i$$

Надо: Найти такой набор значений переменных, чтобы формула приняла значение “истина”.