

Алгоритмы и Структуры Данных

Коченюк Анатолий

22 сентября 2021 г.

Глава 1

Алгоритмы на графах и строках и фане.

1.1 Графы и обход в ширину

кружочки и стрелочки

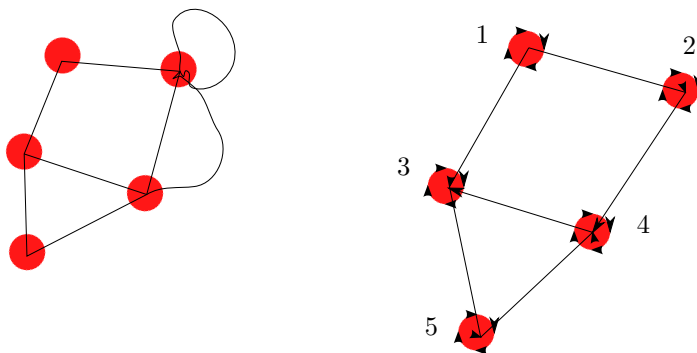


Рис. 1.1: fex

n вершин, m рёбер, $T(n, m)$

Связность – из любой вершины можно прийти до любой другой

$m \geq n - 1$ $m \leq \frac{n(n-1)}{2}$ если связный и нет приколов с кратными рёбрами

и петлями

Определение 1. Матрица смежности – матрица $m(a, b) = \begin{cases} 1, \text{ а и b связаны ребром} \\ 0, \text{ иначе} \end{cases}$

Пример. 1. 2, 3

2. 4

3. 5

4. 2, 3

5. 4

более компактный и полезный способ хранить что с чем связано

Определение 2. Компонента связности – класс эквивалентности по отношению эквивалентности быть связанным.

Определение 3 (Поиск в глубину). Дали нам граф. Берём вершину и помечаем все вершины, которые из неё достижимы. Всё, что мы поместили это кмпонента связности. Дальше берём непомеченную и аналогично выделяем вторую компоненту и так пока вершины не закончатся

```
1  def dfs(v):
2      mark[v] = True
3      for vu in out(v):
4          if !mark[u]:
5              dfs(u)
```

Лемма 1. Мы поместили все достижимые и только их

Доказательство. Ходим только по рёбрам, значит все помеченные вершины достижимы из s

Есть вершина s и достижимая v . Предположим, что мы не дошли. Значит на пути до v была первая вершина, до которой мы не дошли. Но дошли до соседей с ней, значит запустился оттуда и велел непомеченную. Он её пометит в цикле, противоречие. ■

1.2 Что можно делать поиском в глубину в ориентированном графе

Определение 4. Топологическая сортировка – сортировка вершин, чтобы все рёбра шли слева направо

Задача 1. Построить топологическую сортировку у ациклического графа.

Задача 2. В ациклическом графе есть вершина, в которую ничего не входит. Вставим самой левой в сортировке и уберём из графа.

Возьмём любую вершину, в которую ничего не входит. Добавляем в сортировку, убираем из графа.

```
1  z = []
2  for v = 0 .. n-1:
3      if deg[v] == 0:
4          z.insert(v)
5      while !z.empty():
6          x = z.remove()
7          for y in out(x):
8              deg[y]--
9              if deg[y] == 0:
10                 z.insert(y)
```

Время алгоритма $O(m)$

Доказательство.

```
mark[v] = True
2  for m in out(v):
3      if !mark[u]:
4          dfs(u)
5  topsort.add(v)
6
```

■

Утверждение 1. Все рёбра идут слева направо.

Задача 3. Как понять есть ли циклы

Доказательство. Запустить topsort. Если получилась фигня, значит есть цикл. ■

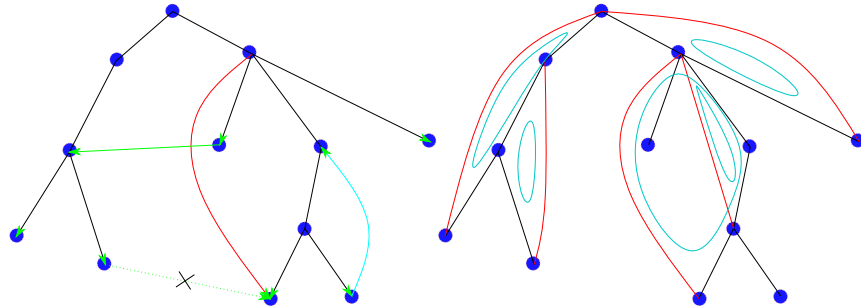


Рис. 1.2: dfstree

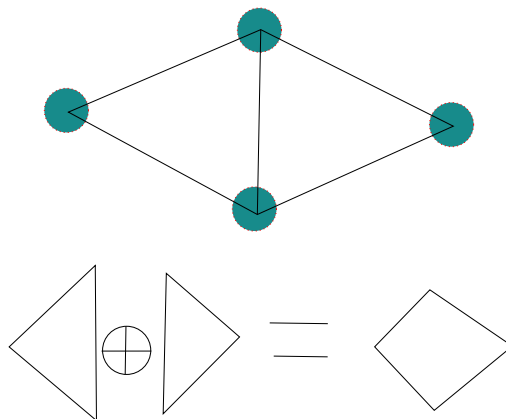


Рис. 1.3: xor

Выберем для всех рёбер, до которых мы не дошли в dfs по циклу из него и рёбер из dfs. Из получившихся циклов можно собрать (ксорами множеств) любой цикл)

1.3 Связность в ориентированных графах

Замечание. Сильная связность является отношением эквивалентности. Можно выбирать классы эквивалентности – компоненты связности. После

этого можно построить конденсацию – граф на компонентах связности, где обозначается односторонняя связь между компонентами.

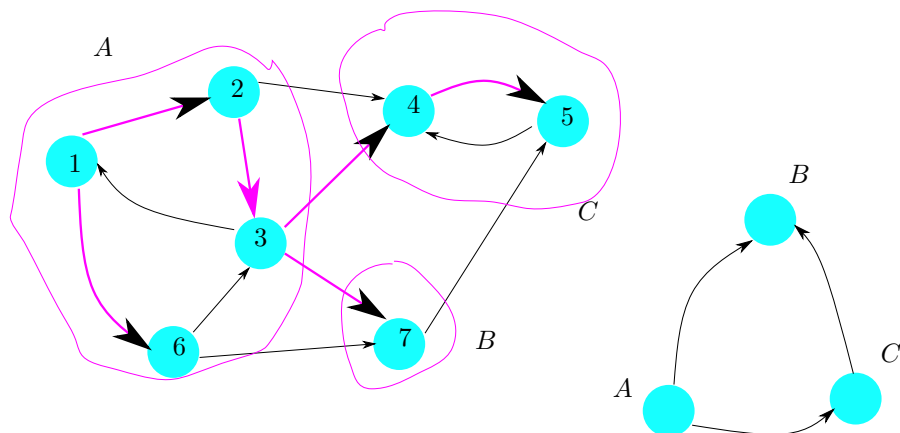


Рис. 1.4: dvureb

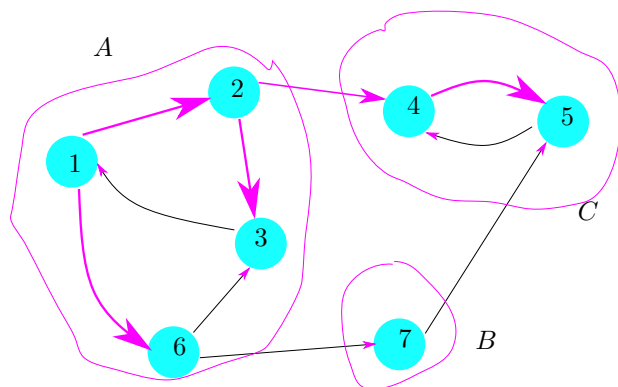


Рис. 1.5: neprimer

Алгоритм:

1. запускаем dfs, записываем вершины в порядке выхода.
2. Если идти по входящим рёбрам из первой вершины в списке, то мы пометим все вершины в компоненте связности 1. Затем перейдя к следующей не помеченной вершине, мы пометим вторую компоненту и

Т.д.

```
1  dfs(v):  
2      ...  
3      p.push_back(v)  
4      -> 1 6 2 3 7 4 5 -- dvureb  
5      -> 1 6 7 2 3 4 5 -- neprimer
```

```
1  for i = 0 .. n-1  
2      dfs1(i)  
3      reverse(p)  
4  for i = 0 .. n-1:  
5      if !mark[p[i]]:  
6          dfs2(p[i])
```

Доказательство. Возьмём компоненту связности. Возьмём первую вершину оттуда, в которую зашёл dfs. Он оттуда не уйдёт, пока всё в ней не пометит.

На компонентах сильной связности есть “топсорт” по первым вершинам в компонентах, которые рассматривает dfs. Это гарантирует нам, что мы будем запускать dfs по обратным рёбрам в компонентах, все входящие компоненты в которую мы уже пометили. Значит dfs2 останется только идти внутри компоненты, что нам и требовалось. ■

Задача 4 (2-SAT). $(x \vee y) \wedge (!y \vee !x) \wedge (z \vee !x) = 1$

$x \vee y = !x \rightarrow y$

Если есть стрелки в обе стороны между x и $\neg x$, то это противоречие.

$A \implies B \iff \neg B \implies \neg A$ – кососимметричность импликации

Если есть пусть между u и v , то есть обратный между $\neg v$ в $\neg u$.

Алгоритм: в конденсации строим топсорт (он ациклический). В каждой паре компонент, ту, которая правее, делаем True.

Доказательство. Берём левую вершину. Все следствия из неё выполняются. Из неё рёбра только выходят. В ней значение False, значит все следствия выполняются. Рассмотрим симметричную к ней, она возможно где-то в середине. В неё только входят рёбра, у неё всё хорошо. Убираем их по индукции всё хорошо. ■

1.4 Двусвязность

Рёберно-двузначный – два пути не пересекающихся по рёбрам

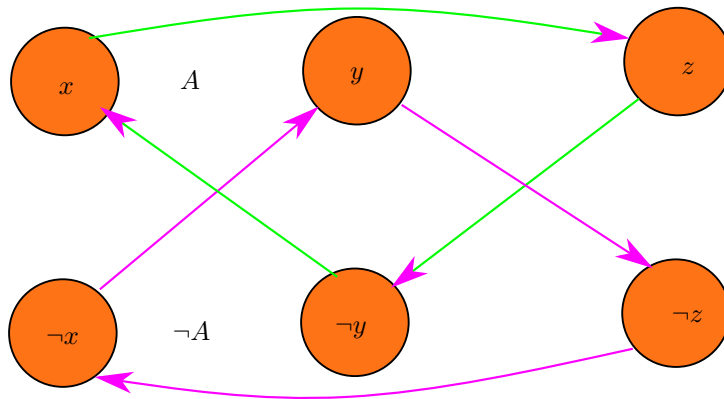


Рис. 1.6: vershinki

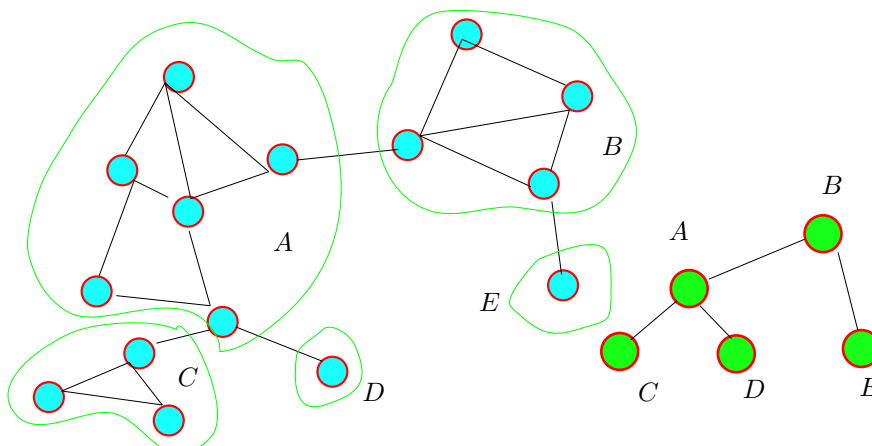


Рис. 1.7: лфлщштшигвкфзр

Утверждение 2. dfs обязательно пройдёт по всем мостам.

Если в поддереве вершины есть рёбро выше неё, то ребро с ней уже не мост.

```

1  dfs(v, p):
2      t_in[v] = T++
3      up[v] = t_in[v]
4      mark[v] = True
5      buf.push(v)
6      for u in out[v]:

```

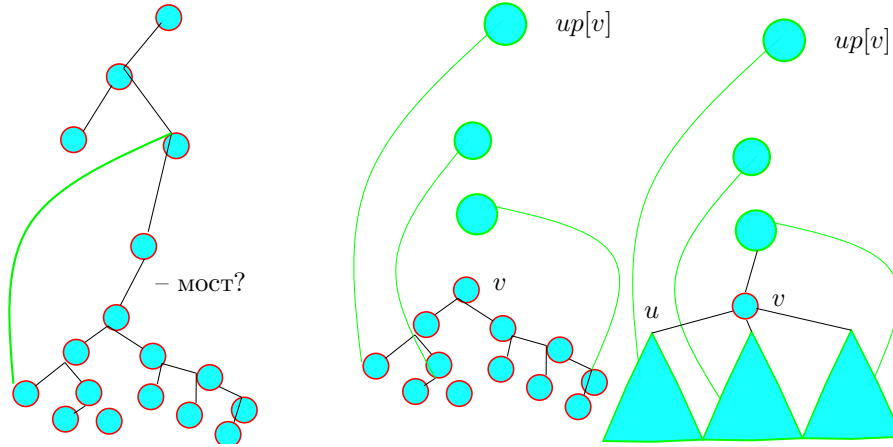


Рис. 1.8: sadfs

```

7         if u == p:
8             continue
9         if not mark[u]:
10             dfs(u, v)
11             up[v] = min(up[v], up[u])
12         else:
13             up[v] = min(up[v], t_in[u])
14         if up[v] == t_in[v]:
15             while True:
16                 x = buf.pop()
17                 comp.add(x)
18                 if x == v:
19                     break

```

Утверждение 3. $\forall u \in \text{child}[v] : up[u] < t_{in}[v] \implies v$ – не точка сочления

верно для всех вершин, кроме корня

```

1     def dfs(v, p):
2         t_in[v] = T++
3         up[v] = t_in[v]
4         mark[v] = True
5         ok = False
6         c = 0
7         for u in out[v]:
8             if u == p:
9                 continue
10            if not mark[u]:
11                dfs(u, v)
12                c++
13            up[v] = min(up[v], up[u])

```

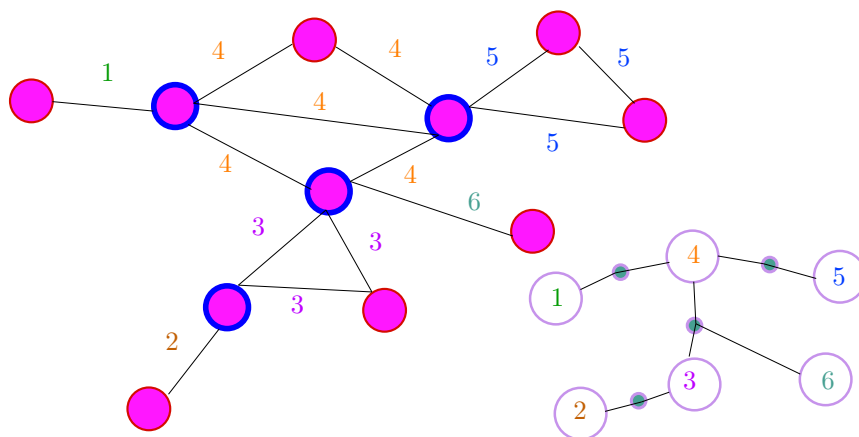


Рис. 1.9: лфкештлгрщсргтфкшыщмфе

```

14         if up[u] >= t_in[v]:
15             ok = True
16         else:
17             up[v] = min(up[v], t_in[u])
18
19     if ok || p = None && c > 1:
20         v - точка сочленения

```

```

1  def dfs(v, p):
2      t_in[v] = T++
3      up[v] = t_in[v]
4      mark[u] = True
5      ok = False
6      for u in out[v]:
7          if u = p:
8              continue
9          if !mark[u]:
10             buf.push(vu)
11             dfs(u,v)
12             up[v] = min(up[v], up[u])
13             if up[u] >= t_in[v]:
14                 while True:
15                     e = buf.pop()
16                     comp.add(e)
17                     if r = (vu):
18                         break
19             else:
20                 up[v] = min(up[v], t_in[u])
21                 if t_in[u] < t_in[v]:
22                     buf.push(vu)

```

Задача 5. Хотим найти эйлеров цикл.

Идём пока идём. Если не идётся, добавляем последнее ребро в цикл и смотрим идётся ли из предыдущей вершины... степени чётные, утыкаемся туда, откуда начали...

для ориентированного графа то же самое, та же логика..

```
1  def dfs(v):
2      vu -- любое непомеченное ребро из v
3      if vu = None:
4          return
5      пометить vu
6      dfs(u)
7      ans.add(vu)
```

Структура данных: умеет проверять пустая ли и брать любой элемент ..
любая структура данных.