Алгоритмы и Структуры Данных

Коченюк Анатолий

18 сентября 2021 г.

Глава 1

Алгоритмы на графах и строках и фане.

1.1 Графы и обход в ширину

кружочки и стрелочки

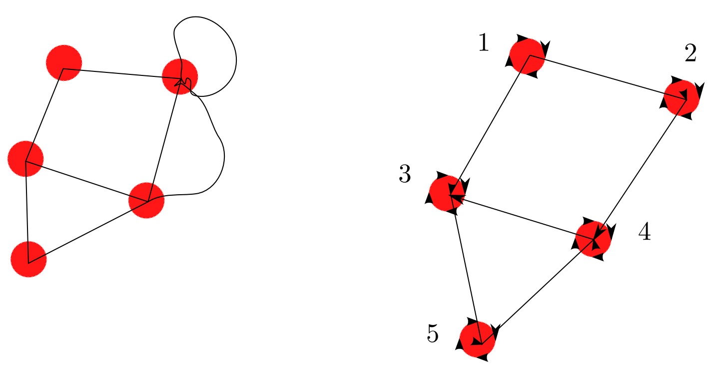

Рис. 1.1: fex

n вершин, m рёбер, $T(n, m)$

Связность – из любой вершины можно дойти до любой другой

$m \geqslant n - 1 \quad m \leqslant \frac{n(n-1)}{2}$ если связный и нет приколов с кратными рёбрами и петлами

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 связаности – 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 пометили это кмпонента связности. Дальше берём непомеченную и аналогично выделяем вторую компоненту и так пока вершины не закончатся

```
dfs(v):
    mark[v] = True
    for vu \in out(v):
        if !mark[u]:
            dfs(u)
```

Лемма 1. Мы пометили 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 соседеней с ней, значит запустился оттуда и ведел 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.

```
z = []
for v = 0 .. n-1:
     if deg[v] = 0
         z.insert(v)
 while !z.empty():
     x = z.remove()
     for y in out(x):
         deg[y]--
         if deg[y] == 0:
             z.insert(y)
```

Время алгоритма $O(m)$

Доказательство.

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

```

■

Утверждение 1. Все рёбра идут слева направо.

Задача 3. Как понять есть ли циклы

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

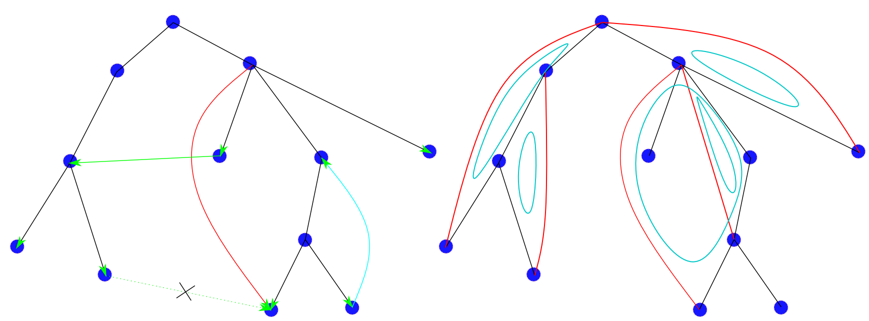

Рис. 1.2: dfstree

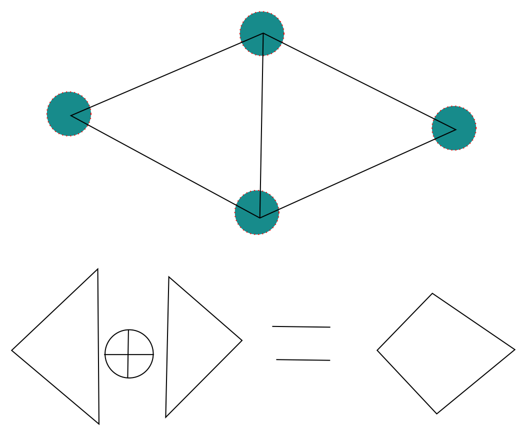

Рис. 1.3: xor

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.

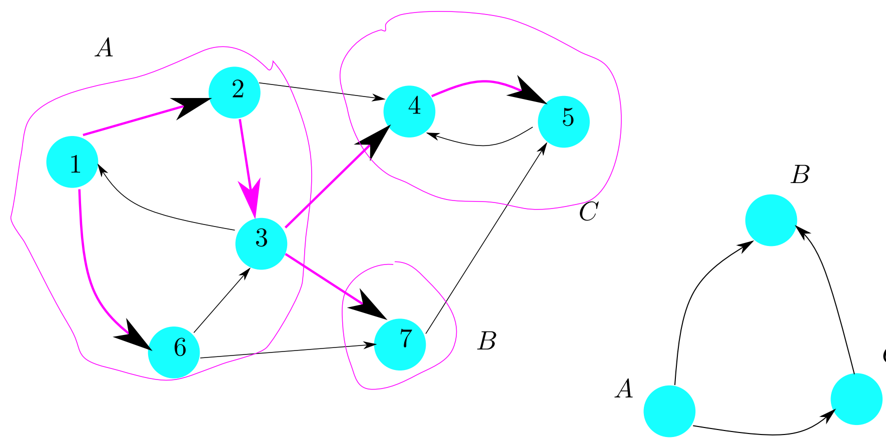

Рис. 1.4: dvureb

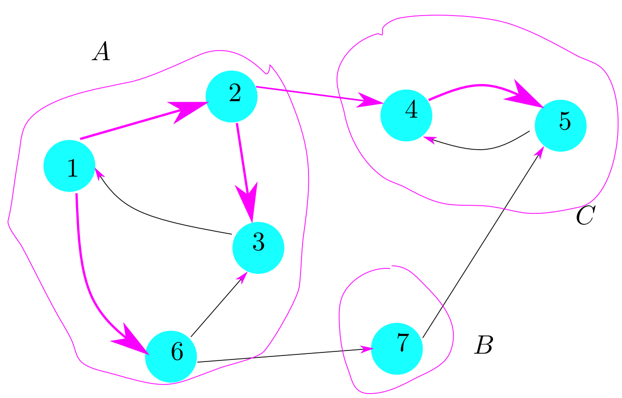

Рис. 1.5: neprimer

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.

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

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

Доказательство. Возьмём компоненту связаности. 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 компоменты в которую мы уже пометили. Значит dfs2 останется только идти внутри компоненты, что нам и требовалось. ■

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

$x \vee y = !x \to y$

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 седствия выполняются. Рассмотрим симметричную к ней, она возможно где-то в середине. В неё только входят рёбра, у неё всё хорошо. Убираем их их по индукции всё хорошо. ■

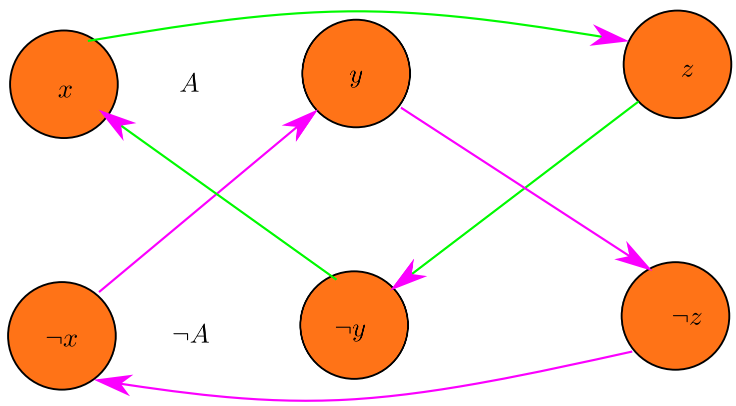

Рис. 1.6: vershinki