Графы

Поиск в глубину (Depth-first search)

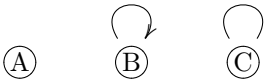
Аминев Б.Д.

October 20, 2017

Изолированная вершина, петля

Изолированной вершиной называется вершина, степень которой равна 0 (A).

Петлёй называется ребро, начало и конец которого находятся в одной и той же вершине (B → B, C → C).



Путь, длина пути, цикл

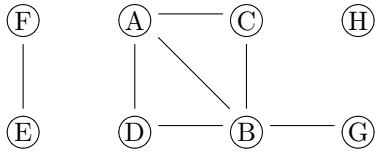
Путь любая последовательность вершин, в которой каждые две соседние вершины соединены ребром ($A \to C \to B \to G$, $A \to D \to B \to D \to B$).

Длина пути количество рёбер в нём (3 и 4 соответственно для примеров выше).

Цикл путь, у которого начальная и конечная вершина совпадают ($A \to C \to B \to D \to A$, $F \to E \to F$).

Простой путь путь, в котором нет повторяющихся рёбер.

Простой цикл цикл, который является простым путём.



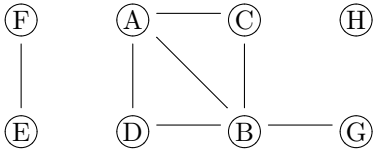
Компоненты связности, связность

Компонента связности неориентированного графа любой набор его вершин, удовлетворяющий свойствам:

- между любыми двумя вершинами набора существует путь;
- набор нельзя расширить, добавив в него ещё хотя бы одну вершину, чтобы при этом осталось верным первое свойство.

Связный граф граф с одной компонентой связности.

Несвязный граф граф, у которого более одной компоненты связности.



Полустепени, кратность рёбер

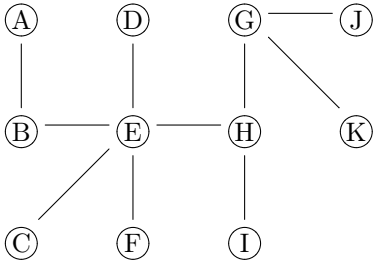
Полустепень захода количество рёбер, входящих в вершину.

Полустепень исхода количество рёбер, исходящих из вершины.

Кратные рёбра рёбра, у которых совпадают начала и концы (с учётом ориентации) — рёбра между A и B, C и D кратны, а между E и F нет.

Дерево

Дерево связный неориентированный граф без петель и кратных рёбер, в котором нет циклов.

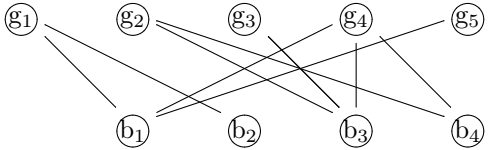


В дереве между любыми двумя вершинами существует единственный простой путь. Для дерева $E = V - 1$.

Висячая вершина вершина степени 1.

Двудольный граф

Граф называется двудольным, если все его вершины можно разбить на две группы, при этом все рёбра должны обязательно идти из одной группы в другую, и запрещены рёбра, соединяющие две вершины из одной группы (b и g).



Алгоритм поиска в глубину

- Алгоритм поиска (или обхода) в глубину (англ. depth-first search, DFS) позволяет построить обход ориентированного или неориентированного графа, при котором посещаются все вершины, доступные из начальной вершины.
- Результатом работы алгоритма является некоторый маршрут, следуя которому можно обойти последовательно все вершины графа, доступные из начальной вершины.
- В каждый момент исполнения алгоритма обрабатывается только одна вершина.
- DFS не находит кратчайших путей, но он применим в ситуациях, когда граф неизвестен целиком, а исследуется каким-то автоматизированным устройством.
- Для ориентированного графа DFS строит дерево путей из начальной вершины во все доступные из нее.

Алгоритм поиска в глубину

1. Пойти в какую-нибудь смежную вершину, не посещенную ранее.
2. Запустить из этой вершины алгоритм обхода в глубину
3. Вернуться в начальную вершину.
4. Повторить пункты 1-3 для всех не посещенных ранее смежных вершин.

Алгоритм поиска в глубину

Сбор информации для дерева обхода в глубину. Реализация на Python

```
visited = [False] * n
prev = [None] * n

def dfs( start, visited, prev, g ):
    visited[start] = True
    for v in g[start]:
        if not visited[v]:
            prev[v] = start
            dfs( v )

dfs( start, visited, prev, g )
```

dfs.py

Алгоритм поиска в глубину

Выделение компонент связности. Реализация на Python

Будем обходить все вершины графа и проверять, была ли очередная вершина посещена ранее. Если не была, то это означает, что найдена новая компонента связности, для выделения всей компоненты связности необходимо запустить DFS от этой вершины.

```
visited = [False] * n

def dfs( start ):
    visited[start] = True
    for v in g[start]:
        if not visited[v]:
            dfs( v )

ncomp = 0
for i in range n:
    if not visited[i]:
        ncomp += 1
        dfs( i )
```

connected_components.py

Алгоритм поиска в глубину

Проверка графа на двудольность. Реализация на Python

```
color = [0] * n
isBipartite = True

def dfs( start ):
  for v in g[start]:
    if color[v] == 0:
      color[v] = 3 - color[start]
      dfs(v)
    else if color[v] == color[start]:
      isBipartite = False

for i in range n:
  if color[i] == 0:
    color[i] = 1
    dfs( i )
```

bipartile.py

Вместо списка visited заведём список color, в котором будем хранить 0 для непосещенных вершин, а для остальных 1 или 2 — их цвет.

Алгоритм DFS для каждого ребра будет проверять цвет его конечной вершины.

Непосещённая вершина красится в цвет, неравный цвету текущей. Если вершина была посещена, то ребро пропускается если его концы разноцветные, инече делается пометка, что граф не является двудольным.

Алгоритм поиска в глубину

Поиск цикла в ориентированном графе. Реализация на Python

```
color = [0] * n
cycleFound = False

def dfs( start ):
  color[start] = 1
  for v in g[start]:
    if color[v] == 0:
      dfs( v )
    elif color[start] == 1:
      cycleFound = True
  color[start] = 2

for i in range n:
  if color[i] == 0:
    dfs( i )
```

cycle.py

Идея: ищем ребро, ведущее из текущей вершины в ту, которая в настоящий момент находится в стадии обработки. Для этого будем красить вершины в три цвета: непосещенные вершины (0), вершины в процессе обработки (1), обработанные вершины (2). Цикл в графе существует, если алгоритм DFS обнаруживает ребро, конец которого покрашен в цвет 1.

Алгоритм поиска в глубину

Топологическая сортировка. Реализация на Python

```
visited = [False] * n
ans = []

def dfs( start ):
    visited[start] = True
    for v in g[start]:
        if not visited[v]:
            dfs( v )
    ans.append( start )

for i in range n:
    if not visited( i ):
        dfs( i )
ans = ans[::-1]
```

topological_sorting.py

Дано: ориентированный граф не содержащий циклов. Вершины графа можно упорядочить так, что все ребра будут идти от вершин с меньшим номером к вершинам с большим номером. Для топологической сортировки графа достаточно запустить алгоритм DFS, при выходе из вершины добавляя вершину в конец списка с ответом. После окончания алгоритма список с ответом развернуть в противоположном порядке.

Алгоритм поиска в глубину

Поиск мостов. Реализация на Python

```
visited = [False] * n
h = [0] * n
f = [0] * n
def dfs( start, parent, curr_h ):
  cur_h += 1
  h[start] = curr_h
  f[start] = h[start]
  visited[u] = True;
  for v in g[start]:
    if v == parent:
      continue
    if not visited[v]:
      dfs( v, start, curr_h )
      f[start] = min( f[start], f[v] )
      if f[v] > h[u]:
        #v-u is bridge!
    else:
      f[start] = min( f[start], h[v] )
```

bridges.py