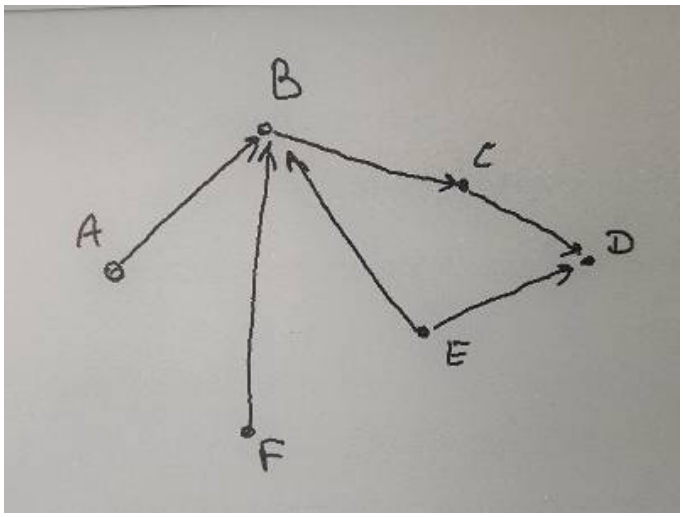
Топологическая сортировка :: Back to task selection
Reward: Ⓝ **0.00**. You performed this task. It is presently accepted.

Пусть нам дан направленный граф без циклов с вершинами V и ребрами E. Назначим каждой вершине в V уникальное число от нуля до $|V| - 1$. Назовем число, назначенное вершине v, s_v. Тогда назначение s называется топологической сортировкой, если для любого ребра из $(u, v) \in E$ верно, что $s_u < s_v$

Например, для следующего графа:



Назначение `A => 0, F => 1, E => 2, B => 3, C => 4, D => 5` является топологической сортировкой, а `A => 0, B => 1, C => 2, D => 3, E => 4, F => 5` не является, потому что, например, для ребра `F -> B` не выполняется условие из определения ($s_F = 5 > s_B = 1$)

Алгоритм нахождения топологической сортировки

Будем обходить граф в глубину, поддерживая глобальный счетчик того, скольким вершинам уже назначено число. При входе в каждую вершину v, сначала рекурсивно вызовем обход в глубину для каждой вершины, в которую есть ребро из v, а затем назначим v следущее число в глобальном счетчике, и уменьшим значение в счетчике на один.

```
given V, E

global counter: int = V.length() - 1;
global visited: bool[]
global assignment: int[]

function dfs(v):
    if visited[v]:
        return
    visited[v] = True
    for w in V such that (v, w) in E:
        dfs(w)
    assignment[v] = counter
    counter -= 1
```

Обход надо вызвать из каждой вершины (не сбрасывая массив посещенных вершин между вызовами):

```
for v in V:
    dfs(v)
```

Корректность алгоритма

Чтобы показать, что итоговое назначение -- это действительно топологическая сортировка, пойдем от противного. Пусть итоговое назначение -- не топологическая сортировка, то есть есть такое ребро (u, v), что `assignment[u] > assignment[v]`. Рассмотрим первый момент, когда `dfs` вошел в вершину u. Прежде чем вершине u, алгоритм сначала рекурсивно спустился во все вершины, в который можно перейти из u по ребру, включая v. При рекурсивном вызове в v могло быть три случая:

1. `visited[v] = True`, и v не на стеке. Тогда когда-то в прошлом `dfs` уже был вызван для v, и завершился, назначив ей какое-то значение. Так как глобальный счетчик только уменьшается, это значение было больше того, которое будет назначено u.
2. `visited[v] = True`, и v на стеке. Тогда есть цикл между u и v, что противоречит требованию, что в графе нет циклов.
3. `visited[v] = False`. Тогда `dfs` выполнится для v, и назначит ей значение, прежде, чем возобновится для u.
Во всех трех случаях есть противоречие, и, следовательно, выполнение dfs из всех вершин находит топологическую сортировку.

Сложность

Сложность алгоритма равна сложности обхода в глубину. Если граф задан списками ребер, то сложность алгоритма $O(|E|)$, а если матрицей смежности, то $O(|V|^2)$.

Task Type: Text Solution Type: Checkboxes

Task

Если на вход DFS выше дать граф с циклами, какие из следующих утверждений будут верны.

Solution

Correct solution

- [ ] Алгоритм может уйти в бесконечный цикл / бесконечную рекурсию
- [ ] Для каких-то вершин строка `assignment[v] = counter` никогда не выполнится
- [x] Для какого-то ребра (u,v) будет верно что `assignment[u] > assignment[v]`
- [ ] Для графа, заданного матрицей смежности, алгоритм может выполняться дольше чем \(O(N^2)\)

Wrong solution (What is this?)

- [ ] Алгоритм может уйти в бесконечный цикл / бесконечную рекурсию
- [x] Для каких-то вершин строка `assignment[v] = counter` никогда не выполнится
- [ ] Для какого-то ребра (u,v) будет верно что `assignment[u] > assignment[v]`
- [x] Для графа, заданного матрицей смежности, алгоритм может выполняться дольше чем \(O(N^2)\)

Explanation:
Заметим, что за исключением последних двух строк, реализация dfs -- это классическая реализация, и для такой реализации известно что (а) она завершается за O(N^2) для графа, заданного матрицей смежности, и (б) посещает каждую вершину ровно по одному разу. Это исключает варианты 1, 2 и 4.
Вариант 3 истинен, потому из (б) выше следует, что каждая вершина будет посещена, и следотвательно каждой вершине v будет назначено какое-то значение в assignment[v], более того, так как счетчик только уменьшается, это значение для каждой вершины уникально. Для ребер на цикле не может быть, что для каждого ребра (u, v) `assignment[u] < assignment[v]`, так как отношение "меньше" транзитивно, и так как u достижима из себя по циклу, из этого следовало бы что `assignment[u] < assignment[u]`.

Task Type: Photo / Picture Solution Type: Text

Task Description: **Для графа ниже была найдена некоторая топологическая сортировка. Какое значение может быть в assignment для вершины D? Укажите в порядке возрастания через запятую без пробелов**

Correct solution

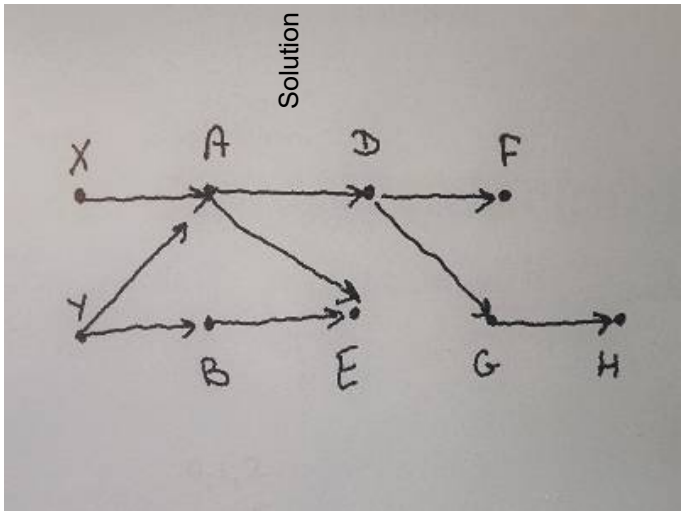
3,4,5

Wrong solution (What is this?)

4,5

Typesetting math: 100%

Solution

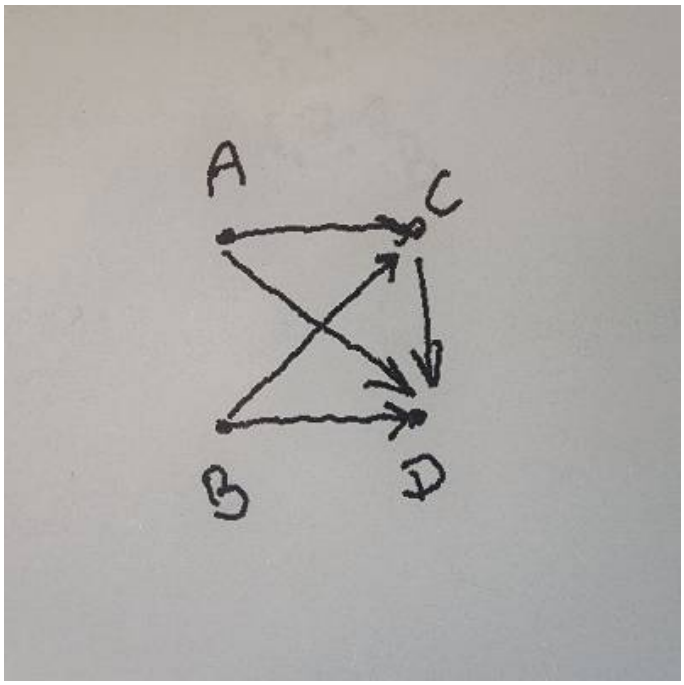


Explanation:
Всего в графе 9 вершин, и они получат значения от 0 до 8 включительно. Вершина D не может иметь значения 0, 1 и 2, потому что вершины A, X и Y должны иметь значения меньше чем она. Аналогично, вершины F, G и H должны иметь значения выше чем у D, и следовательно значение D не может быть 6, 7 и 8. Это оставляет 3, 4, 5.

Task Type: Photo / Picture Solution Type: Checkboxes

Task Description: **Какие из назначений являются топологическими сортировками графа на картинке?**

Task



Solution

Correct solution
- [x] A => 0, B => 1, C => 2, D => 3
- [ ] A => 3, B => 2, C => 1, D => 0
- [ ] A => 0, B => 2, C => 1, D => 3
- [ ] A => 1, B => 1, C => 2, D => 3
- [x] A => 1, B => 0, C => 2, D => 3

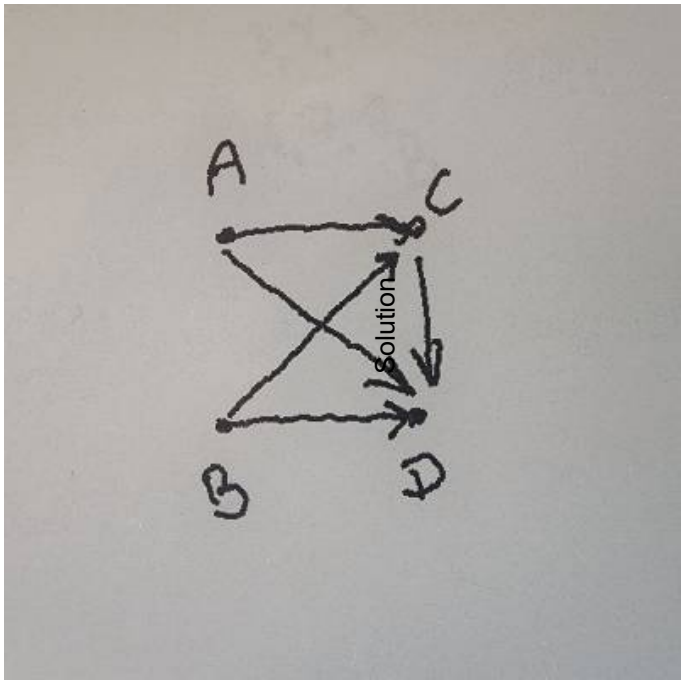
Wrong solution (What is this?)
- [x] A => 0, B => 1, C => 2, D => 3
- [x] A => 3, B => 2, C => 1, D => 0
- [ ] A => 0, B => 2, C => 1, D => 3
- [ ] A => 1, B => 1, C => 2, D => 3
- [ ] A => 1, B => 0, C => 2, D => 3

Explanation:
Всего есть пять ребер A -> B, B -> C, A -> D, C -> D, B -> D
Первый пример назначет вершинам значения в порядке алфавита. Так как все ребра идут из более ранней буквы алфавита в более позднюю, это является правильной топологической сортировкой.
Во втором примере инвариант нарушен, например, для ребра A -> C.
В третьем примере инвариант нарушен для ребра B -> C.
В четвертом примере двум вершинам назначено одинаковое число (1).
Пятый пример -- правильная топологическая сортировка. По сравнению с первым примером поменяли местами значения для A и B (которые были 0 и 1). Так как в A и B нет входящих ребер, а для всех исходящих значеня выше чем 1, это по прежднему топологическая сортировка.

Task Type: Photo / Picture Solution Type: Text

Task Description: **Сколько различных назначений чисел от 0 до 3 вершинам графа ниже являются сорировками?**

Typesetting math: 100%



Solution

Correct solution
2

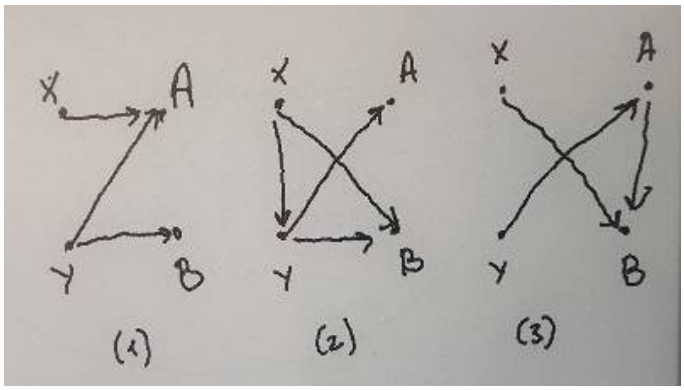
Wrong solution (What is this?)
4

Explanation:
Заметим, что вершина D достижима из всех остальных вершин, и, следовательно, должна иметь значение 3.
Вершина C достижима из A и B, и должна иметь значение 2.
A и B не связаны ребром, и могут быть либо 0 и 1 соответственно, либо 1 и 0.

Task Type: Photo / Picture Solution Type: Text

Task Description: **Для каждого из трех графов ниже нашли некоторую топологическую сортировку, а затем поменяли в ней значения, назначенные A и B. Определите, верно ли, что новое назначение -- тоже топологическая сортировка. Для каждого графа ответ должен быть "да", "неизвестно" или "нет". Ответы для графов слева направо выведите через запятую без пробелов (например да,да,нет).**

Task



Solution

Correct solution
неизвестно,да,нет

Wrong solution (What is this?)
да,да,нет

Explanation:
Для третьего графа ответ однозначно нет, потому что есть ребро из A в B, и если до изменения инвариант выполнялся, то после изменения он не выполняется.
Для второго графа есть только две топологических сортировки: X => 0, Y => 1, A => 2, B => 3 и X => 0, Y => 1, A => 3, B => 2. Они отличаются тем, что у A и B меняются местами значения.
Для первого графа значение в B может быть меньше чем значение в X, и тогда при изменении назначений для A и B нарушится инвариант для ребра X -> A.

Typesetting math: 100%