ГРАФЫ
ПОИСК ЦИКЛОВ
ОПРЕДЕЛЕНИЕ ПРЕДКОВ В ДЕРЕВЕ

Школа::Кода

Олимпиадное программирование

2020-2021 Таганрог

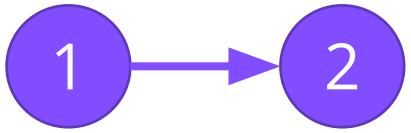
Определение

- Предок (в наиболее используемом понимании), родитель – это вершина, из которой был совершён переход в данную.
- Предок вершины в дереве – любая вершина, расположенная на пути от данной до корня.
- Наименьший общий предок двух вершин – это вершина максимально удалённая от корня дерева, которая является предком для обоих этих вершин.

Алгоритм поиска циклов

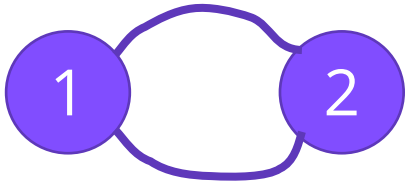
1. Проведём поиск в глубину. Если граф ненаправленный, то дополнительно для каждой вершины будем хранить родителя (номер вершины, из которой мы пришли в текущую), переход в родителя осуществлять не будем.
2. Если в какой-то момент обхода мы зашли в серую вершину, то мы нашли цикл.
3. С момента нахождения цикла будем добавлять текущую вершину в перечень вершин цикла и откатываться к предыдущей, пока не дойдём то старта цикла.

Разница в нахождении цикла в направленном и ненаправленном графах



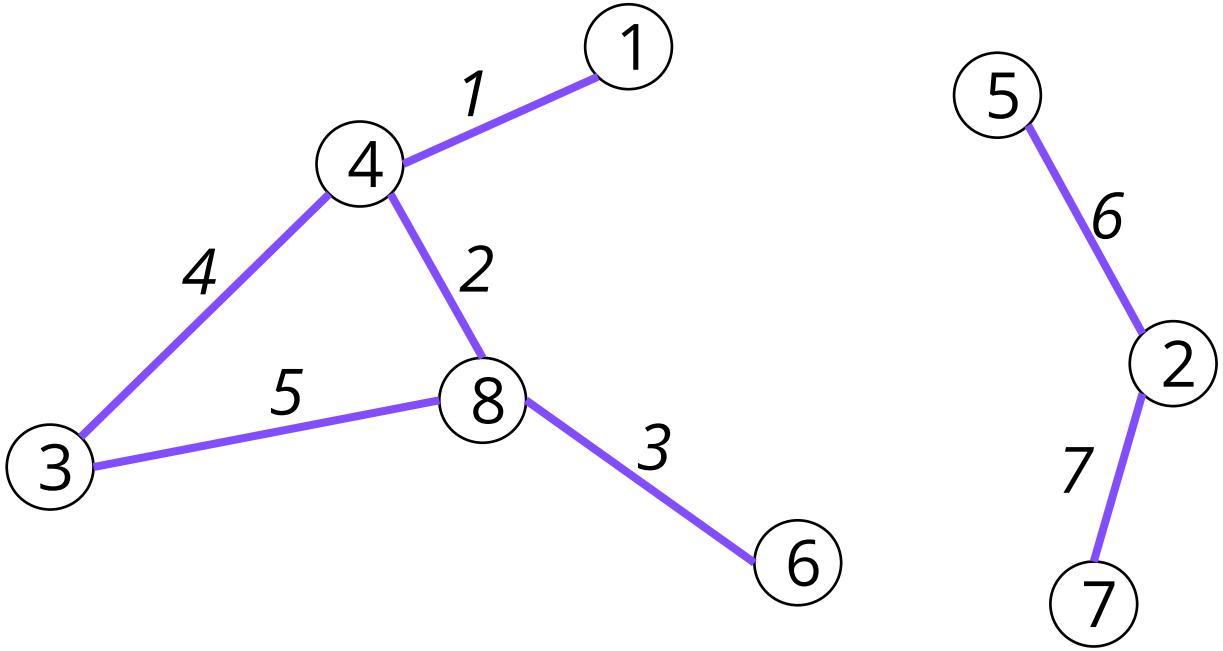
Нет цикла

Нет цикла



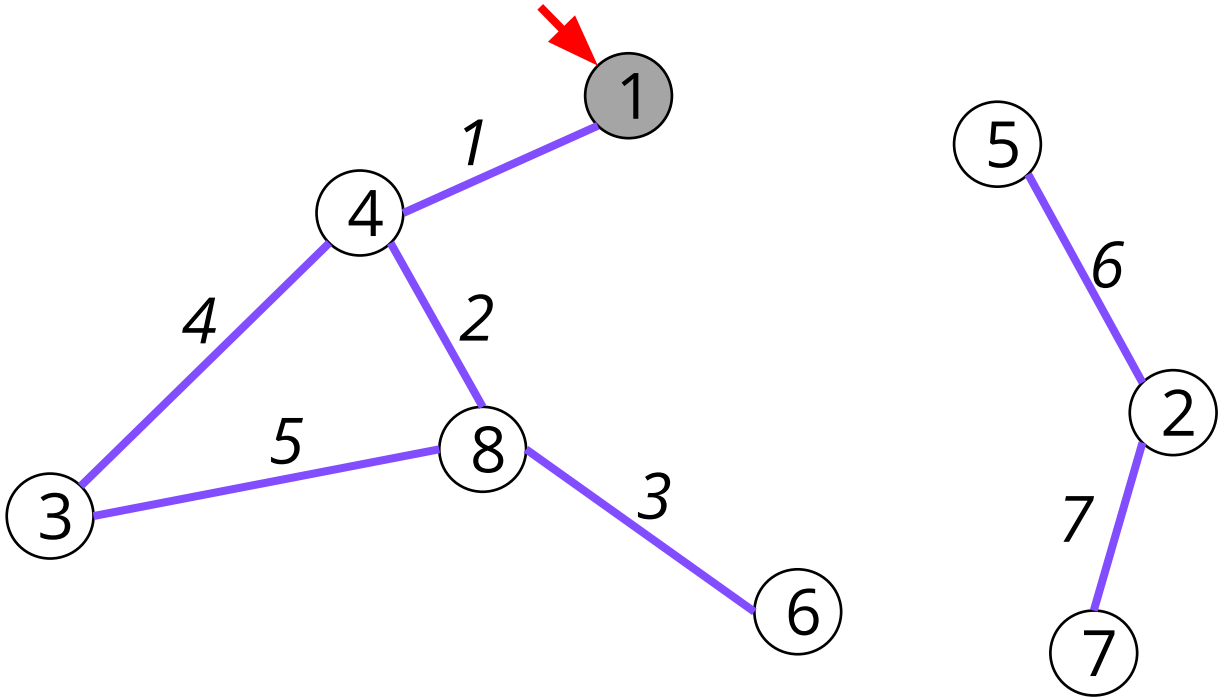
Есть цикл

Поиск циклов. Пример. Шаг 0



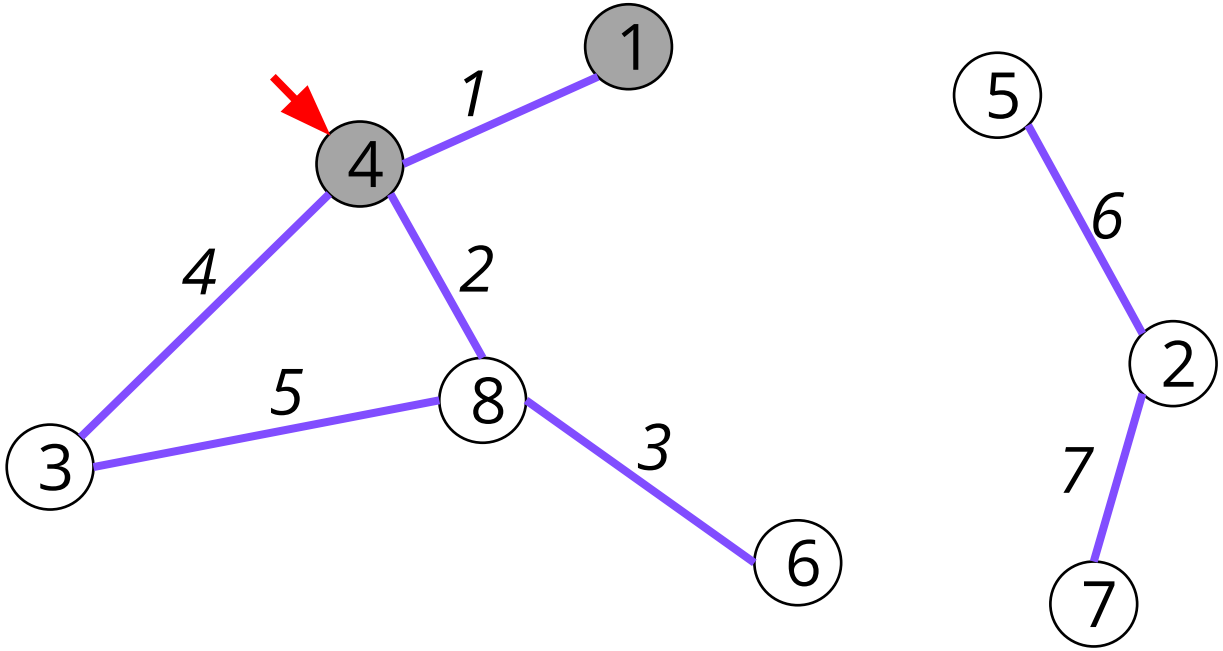
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 1



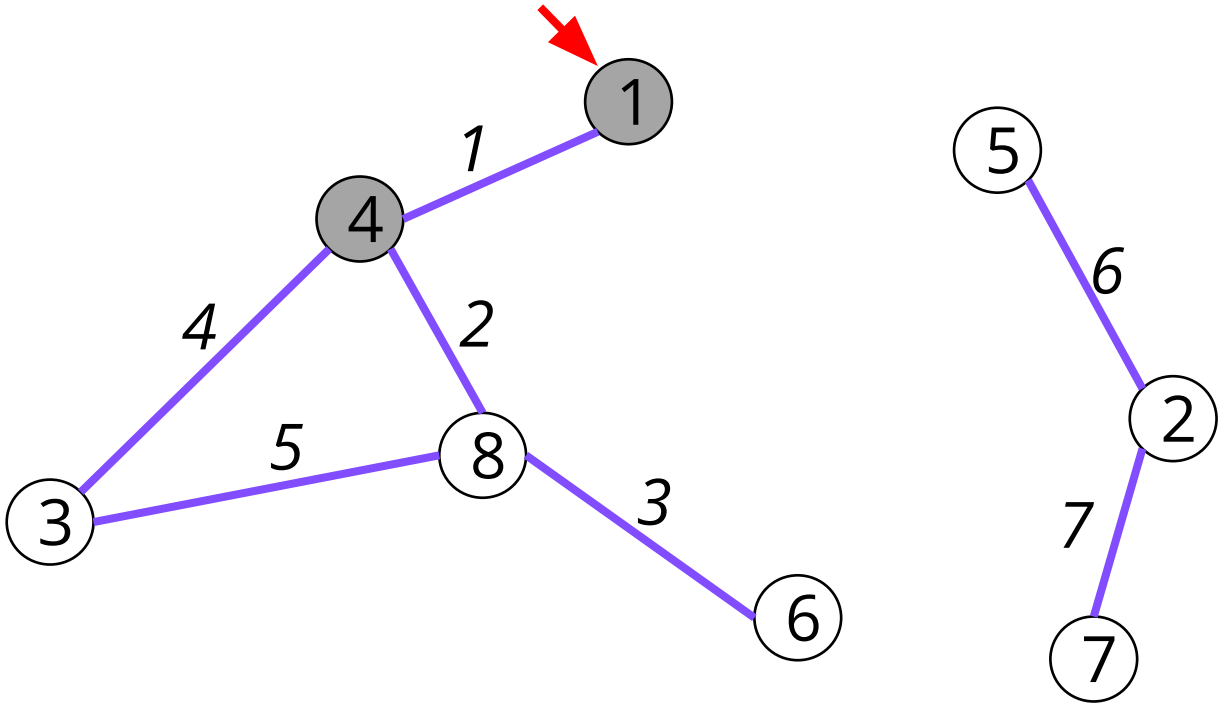
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 2



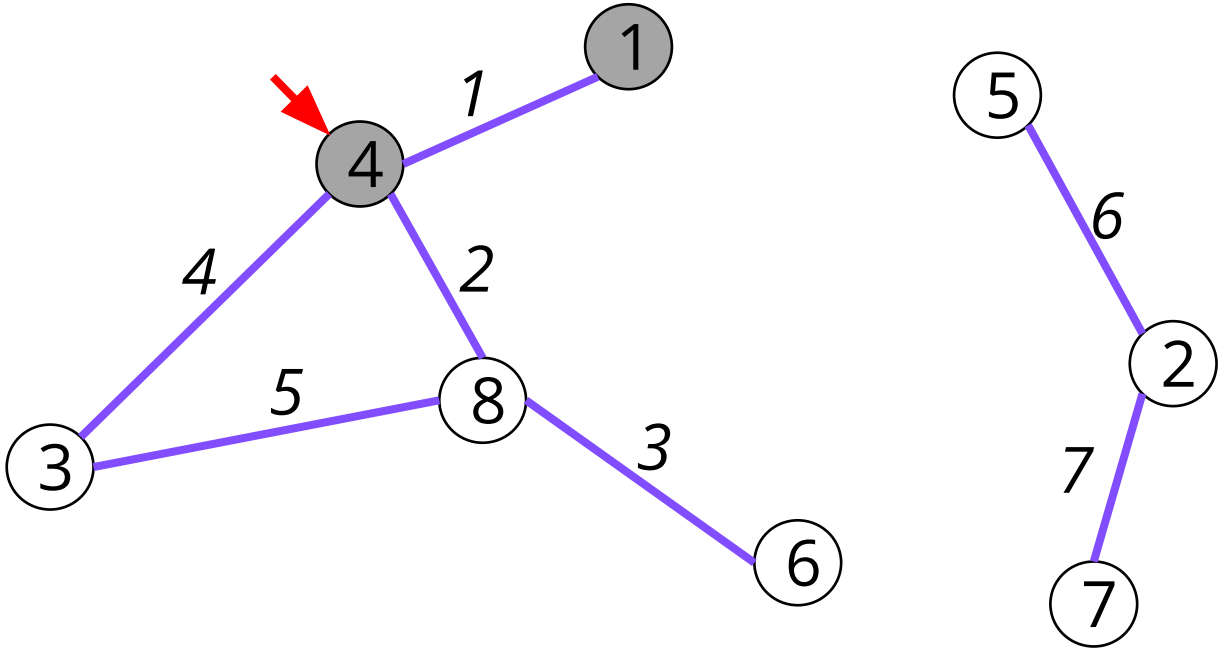
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 3



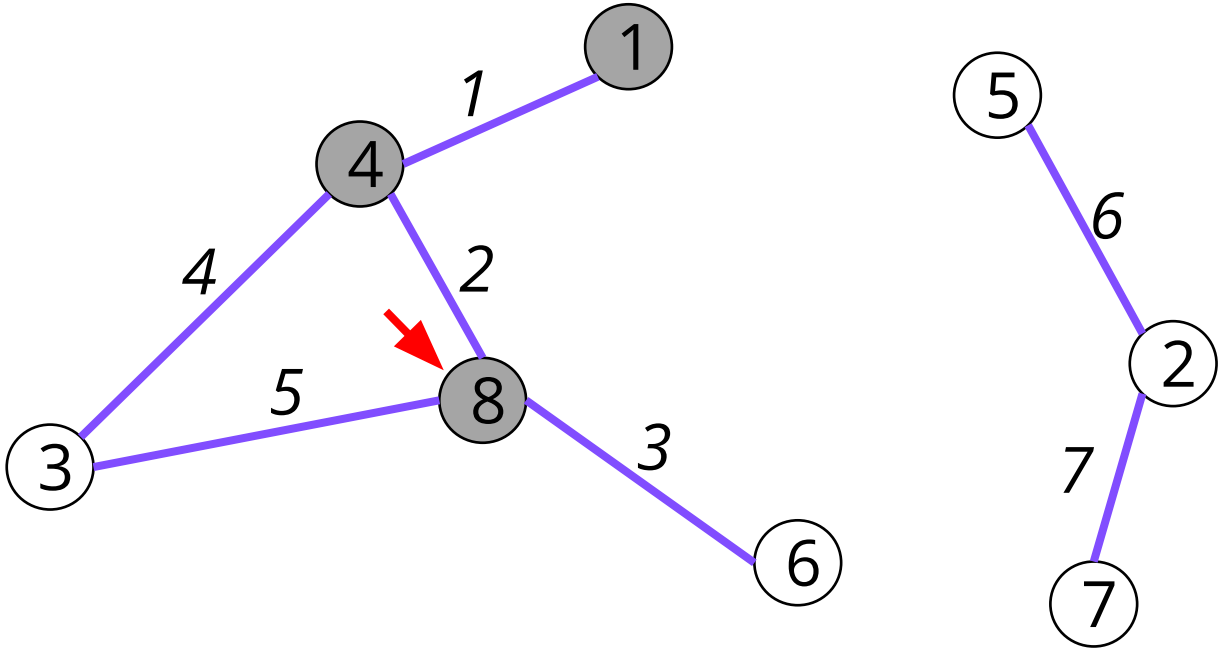
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 4



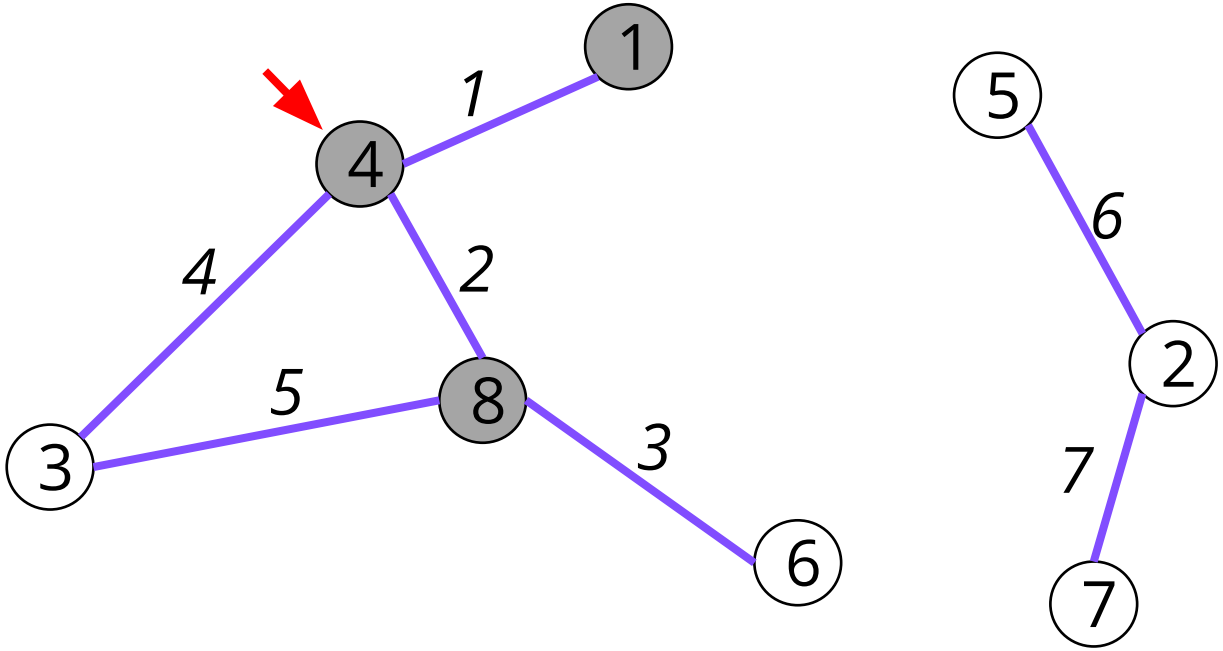
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 5



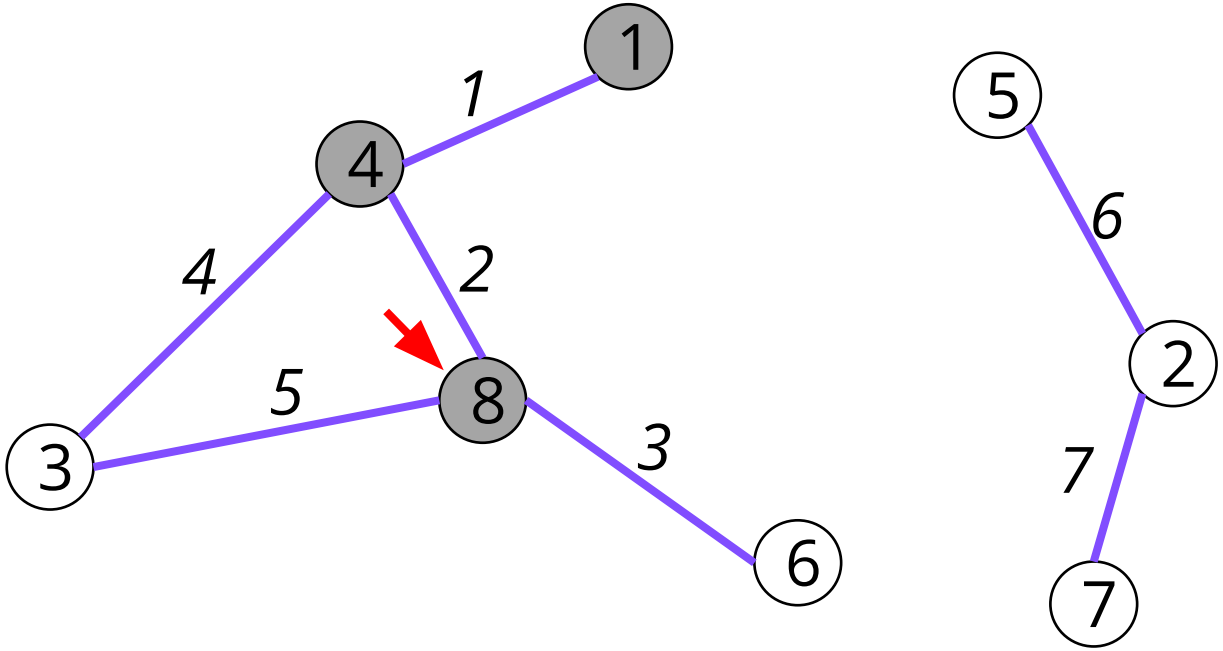
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 6



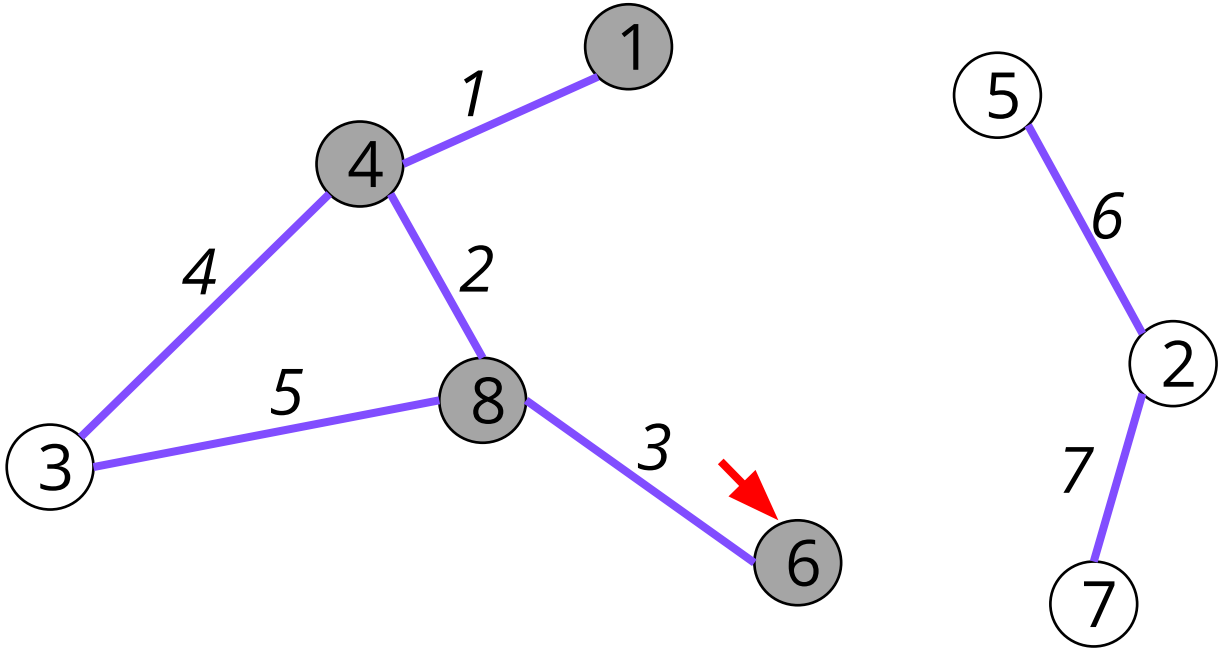
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 7



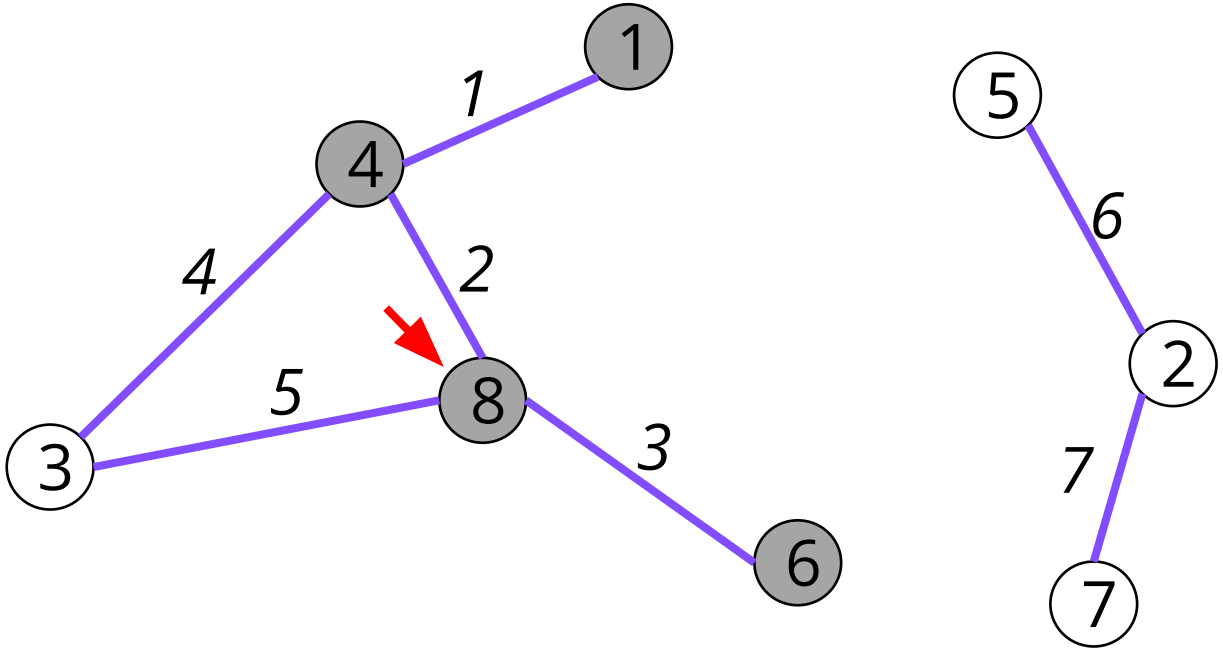
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 8



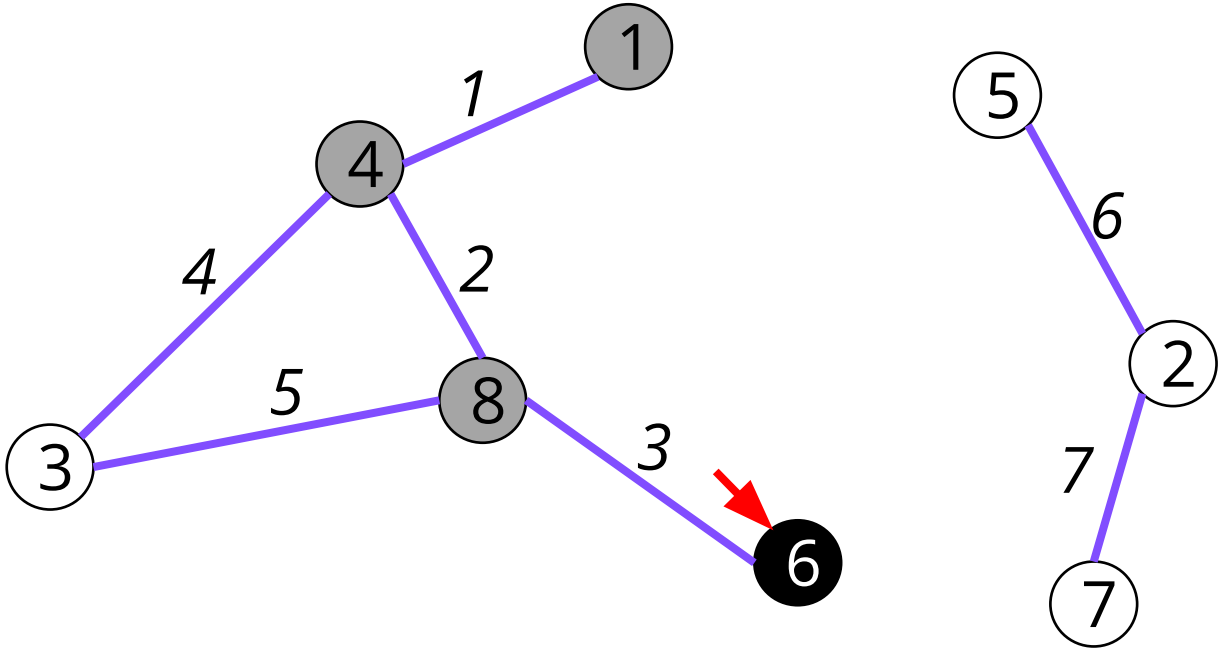
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 9



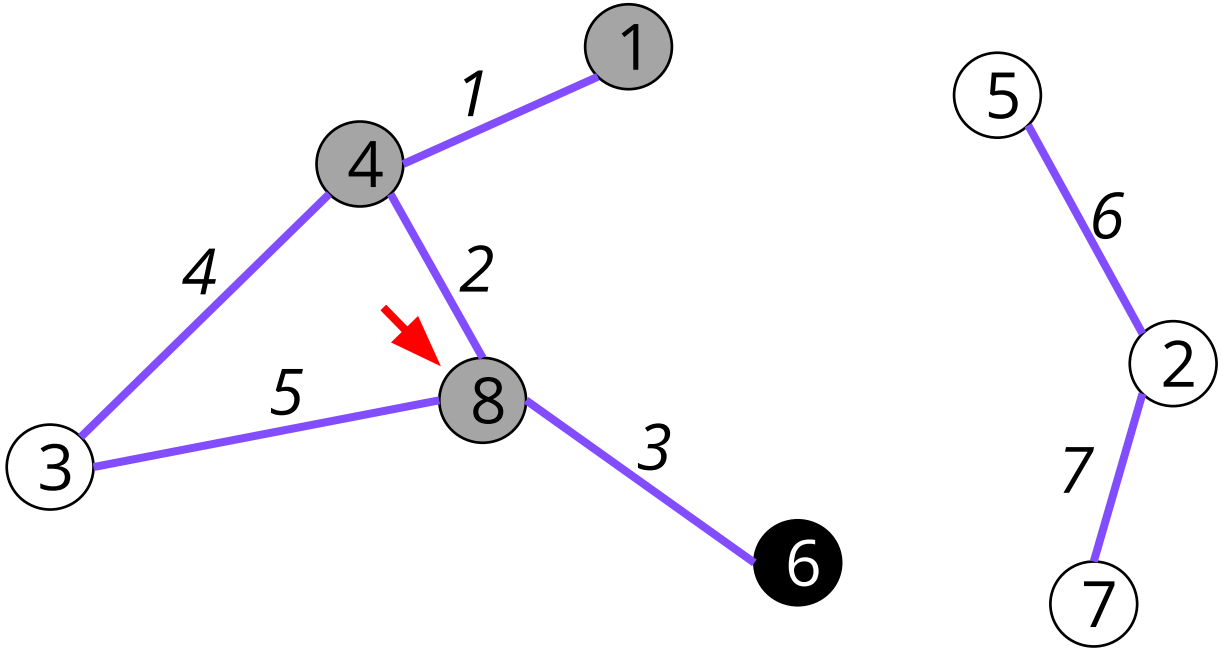
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 10



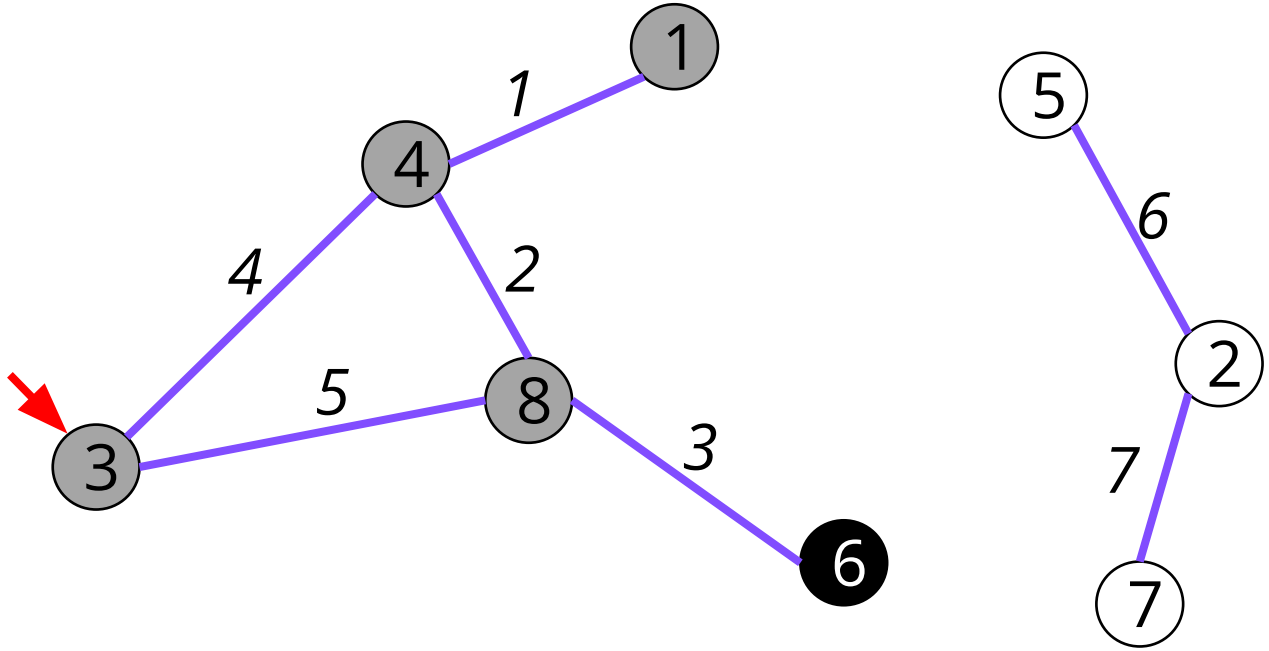
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 11



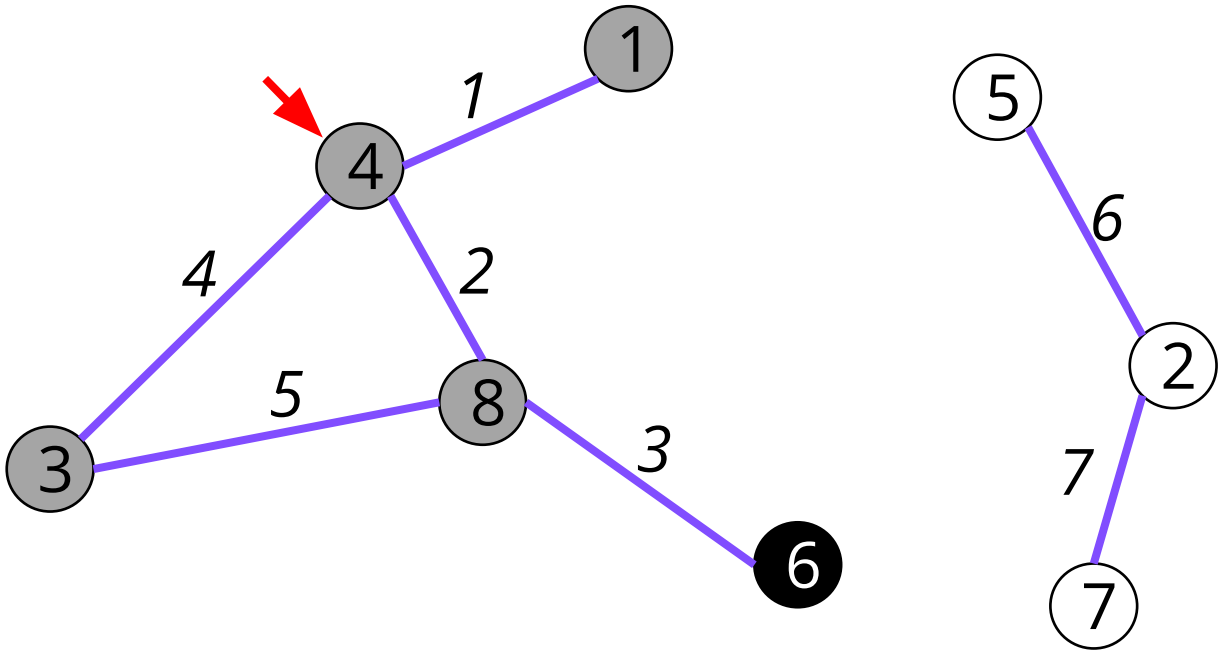
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 12



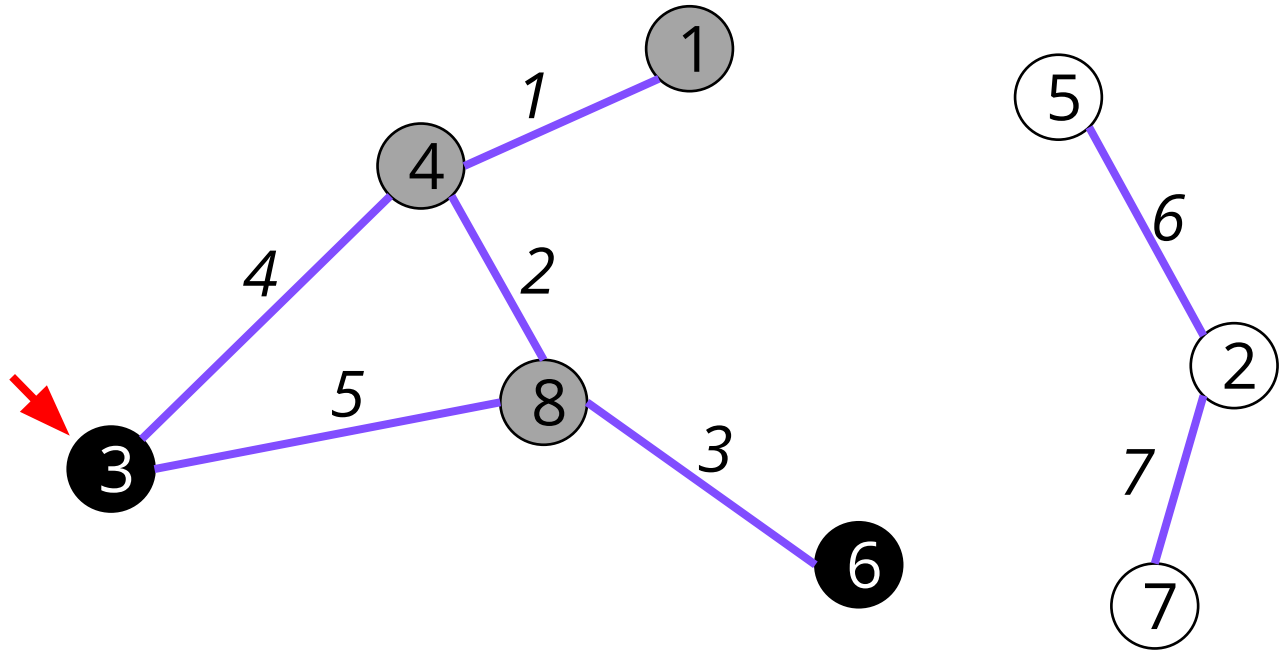
Цикл найден – нет
Вершины в цикле:

Поиск циклов. Пример. Шаг 13



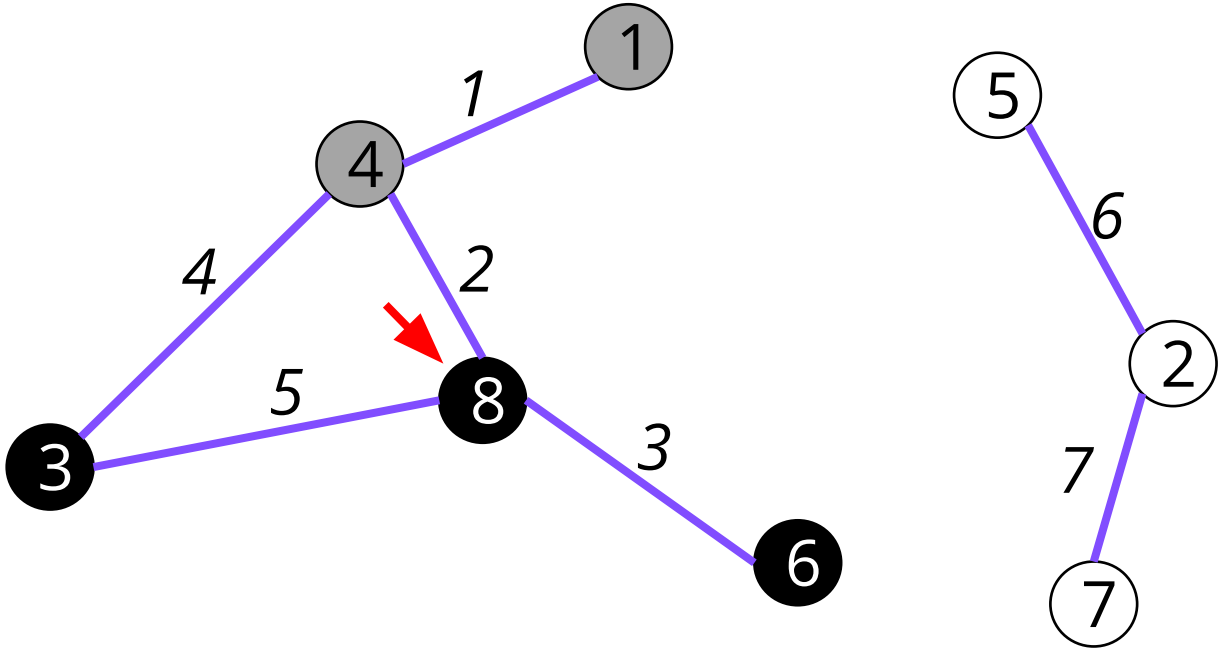
Цикл найден – да
Вершины в цикле: 4

Поиск циклов. Пример. Шаг 14



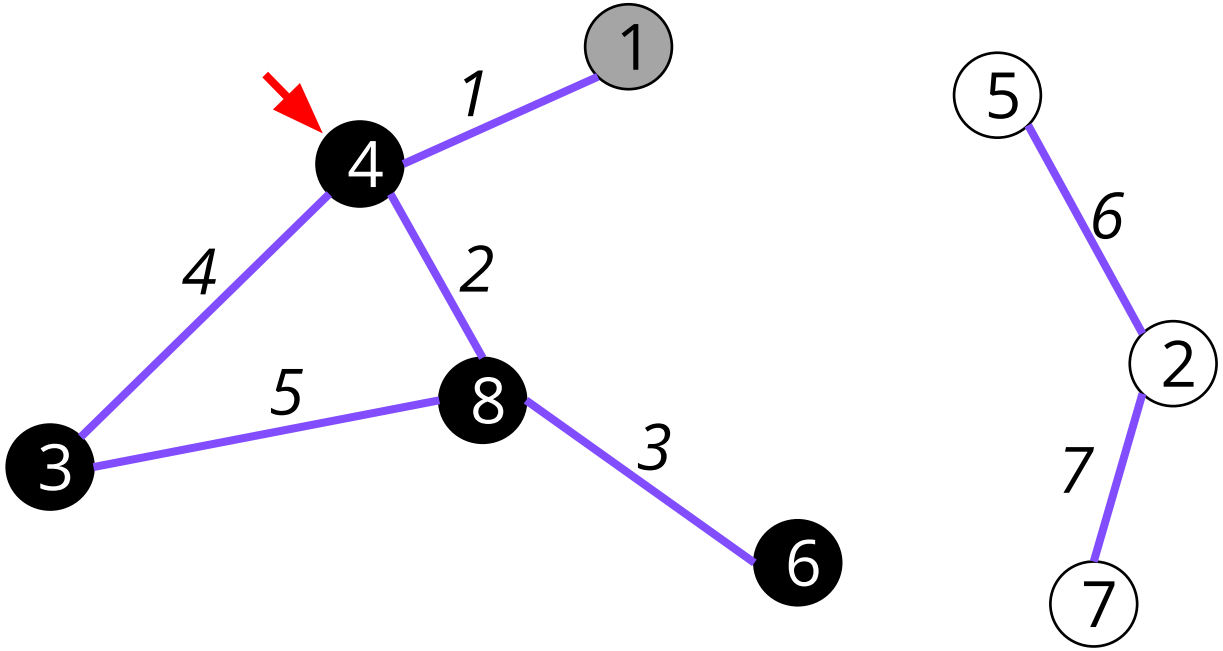
Цикл найден – да
Вершины в цикле: 4, 3

Поиск циклов. Пример. Шаг 15



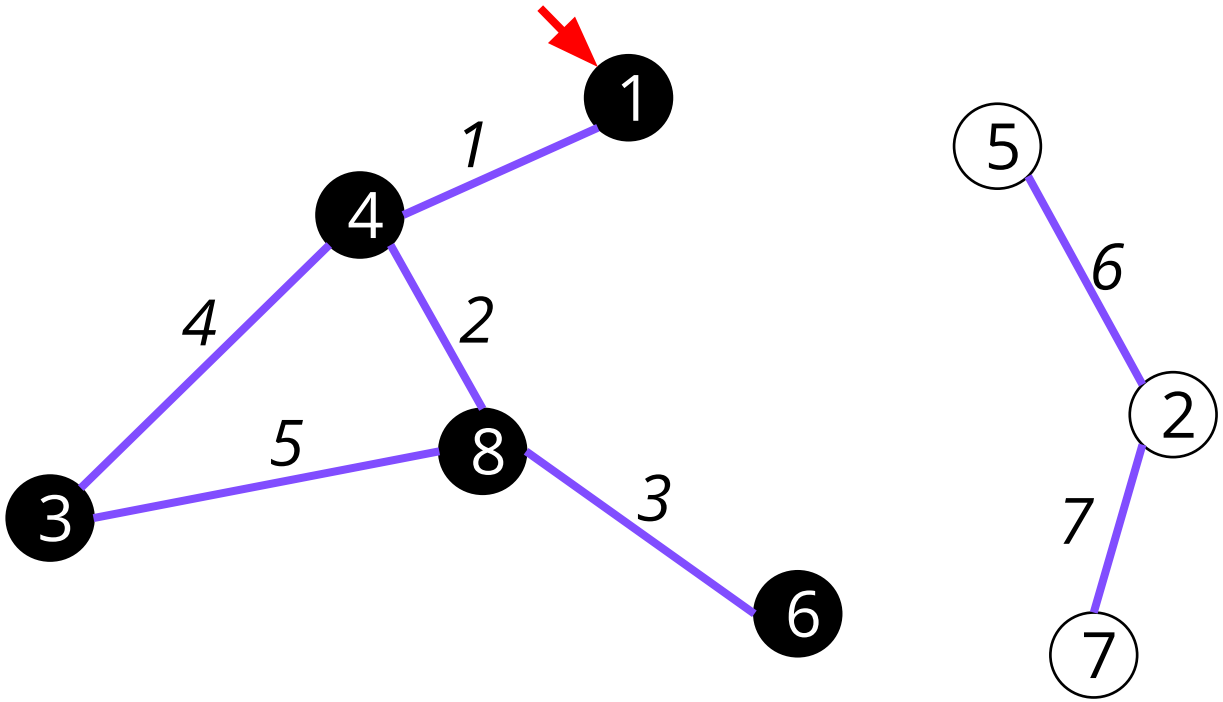
Цикл найден – да
Вершины в цикле: 4, 3, 8

Поиск циклов. Пример. Шаг 16



Цикл найден – да
Вершины в цикле: 4, 3, 8

Поиск циклов. Пример. Шаг 17



Цикл найден – да
Вершины в цикле: 4, 3, 8

Реализация

- На рисунке представлен алгоритм нахождения цикла в ненаправленном графе.
- В цикле в функции main вызов поиска осуществляется как dfs(i, -1), а не dfs(i).
- Для направленного графа не нужно обрабатывать случай с предком.

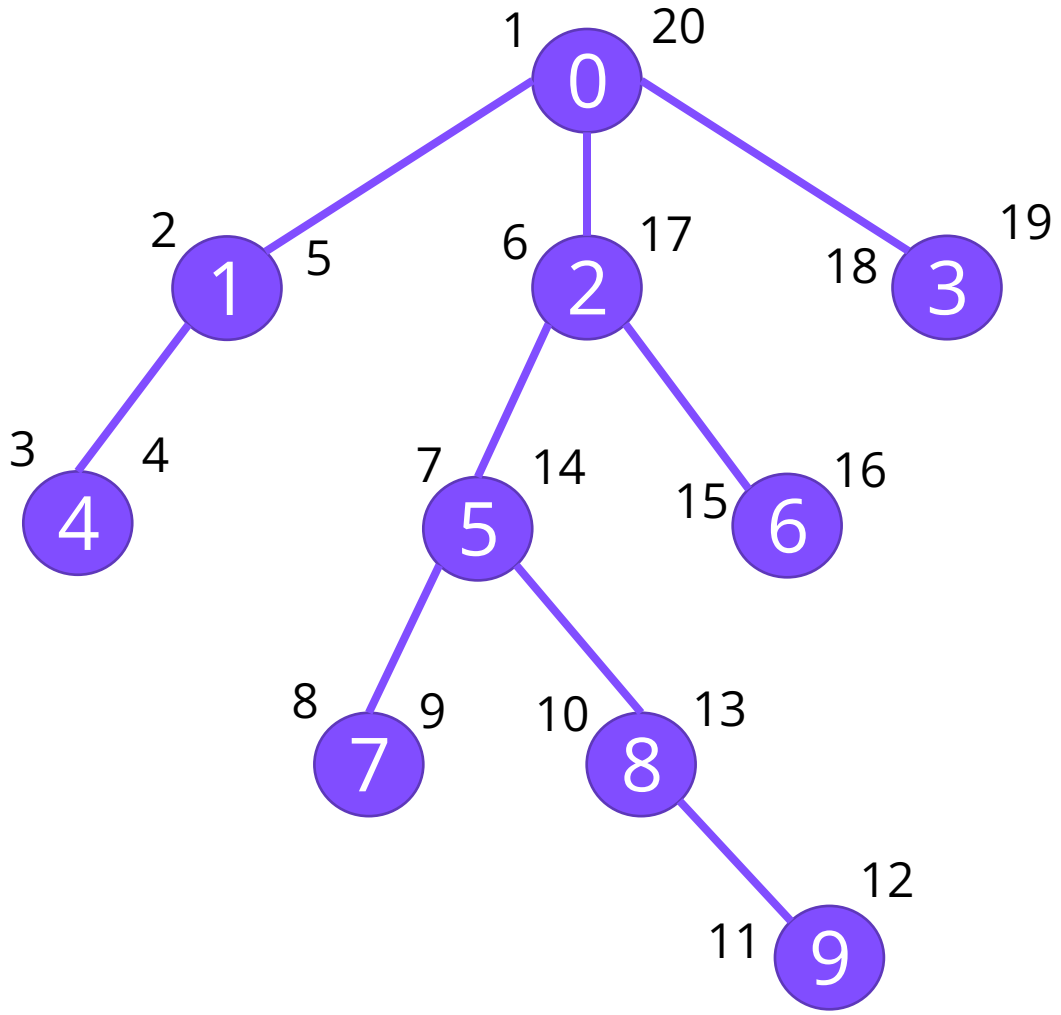
```
vector<int> cycle;
bool flag = false;

bool dfs(int v, int p)
{
    if (g[v].color == 1)
    {
        flag = true;
        cycle.push_back(v);
        return true;
    }
    if (g[v].color == 2)
        return false;
    g[v].color = 1;
    for (int i = 0; i < g[v].to.size(); ++i)
        if(g[v].to[i] != p)
            if (dfs(g[v].to[i], v))
            {
                if (v == cycle[0])
                    flag = false;
                if (flag)
                    cycle.push_back(v);
                return true;
            }
    g[v].color = 2;
    return false;
}
```

Алгоритм проверки является ли одна вершина предком для другой

- Для каждой вершины будем дополнительно хранить время входа в неё (tin) и время выхода (tout).
- Для этого заведём глобальный таймер. При входе в вершину запомним значение таймера в tin, и увеличим таймер на 1. Аналогично при выходе из вершины.
- Вершина A является предком вершины B тогда и только тогда, когда $A.tin \leq B.tin$ и $B.tout \leq A.tout$.
- Стоит отметить, что при такой формулировке каждая вершина является своим же предком. Если такое явление нежелательно, то неравенства следует заменить на строгие.

Предки в дереве



Наивный алгоритм нахождения наименьшего общего предка вершин A и B

1. С помощью обхода в глубину, запущенного из корня, для каждой вершины посчитаем её время входа, время выхода и глубину (удалённость от корня).
2. В качестве результата на старте алгоритма возьмём корень дерева.
3. Переберём все вершины. Если вершина является предком для обеих вершин A и B, и она расположена глубже, чем текущая вершина, хранящаяся в результате, то данная вершина становится новым результатом.

Реализация

```
struct Node
{
    vector<int> to;
    int color = 0;
    int depth;
    int tin;
    int tout;
};
vector<Node> g;

int timer = 0;
void dfs(int v, int d)
{
    if (g[v].color)
        return;
    g[v].color = 1;
    g[v].depth = d;
    g[v].tin = timer++;
    for (int i = 0; i < g[v].to.size(); ++i)
        dfs(g[v].to[i], d + 1);
    g[v].tout = timer++;
}

bool is_parent(Node& a, Node& b)
{
    return a.tin <= b.tin && b.tout <= a.tout;
}
```

```
int main()
{
    int n, a, b;
    cin >> n >> a >> b;
    g.resize(n);
    --a; --b;
    for (int i = 1; i < n; ++i)
    {
        int p;
        cin >> p;
        --p;
        g[p].to.push_back(i);
    }
    dfs(0, 0); //Корень в вершине 0
    int res = 0;
    for (int i = 0; i < n; ++i)
        if (is_parent(g[i], g[a]) &&
            is_parent(g[i], g[b]) &&
            g[i].depth > g[res].depth)
            res = i;
    cout << res + 1 << endl;

    return 0;
}
```