Графы. Часть 1.

Графом называется пара $G = (V, E)$, где V - множество вершин графа, а E - множество ребер.

Графы бывают **неориентировнаными** и **ориенированными**.

Графы бывают **взвешанными** и **невзвешанными**.

Степенью вершины называется количество ребер, выходящих из данной вершины.

Путем называется последовательность вершин v_1,v_2,v_3...v_n, где любыме вершинами v_i $v_i + 1$ связаны ребром в графе. **Длина пути** - количество ребер на нем. Для взвешенного графа весом пути будет являться сумма весов всех ребер на нем.

Циклом называется путь, у которого v_1=v_n. **Простым циклом** называется цикл, в котором все вершины кроме v_n (по определению v_1=v_n) различны.

Неориентированный граф называется **связным**, если из любой вершины существет путь до любой другой вершины. **Компонента связности** - подмножество вершин в графе, такое что они являются свзяными, и ни из одной вершины помножества не существет ребра, ведущего в вершину, непринадлежщую подмножеству.

Дерево - связный неориентированный граф без циклов.

Теорема. Связный неориентированный граф из n вершин и $n-1$ ребра является деревом.

Граф, в котором между любыми парами вершин существует ребро, - называется полным.

Теорема. Количество ребер в полном графе n * (n - 1) / 2.

Путь из a в b называется **кратчайшим**, если не существует пути меньшей длины из вершины a в вершину b.

Способы хранения графа

Обычно во входных данных граф задается количеством вершин n, количеством ребер m и списком всех ребер. Каждое ребро задается 2 числами - парами вершин, которые оно соединяет. Рассмотрим разные способы хранения графа и его считывания. Существует 2 способа хранения графа:

1) Матрица смежности

```
int n, m;
cin >> n >> m;
vector<vector<int> > G(n, vector<int>(n, 0));
for(int i = 0; i < m; ++i)
{
    int a, b;
    cin >> a >> b;
    G[a - 1][b - 1] = 1;
    G[b - 1][a - 1] = 1;
}
```

2) Список смежности

```
int n, m;
cin >> n >> m;
vector<vector<int> > G(n);
for(int i = 0; i < m; ++i)
{
    int a, b;
    cin >> a >> b;
    G[a - 1].push_back(b - 1);
    G[b - 1].push_back(a - 1);
}
```

Второй способ хранения графа является более предпочтительным.

Обход в глубину

Обход в глубину является одним из способов обхода графа.

```
vector<vector<int> > G;
vector<int> color;// color.assign(n, 0) in main
void DFS(int v)
{
    color[v] = 1;
    for (int i = 0; i < G[v].size(); ++i)
    {
        if (color[G[v][i]] == 0)
            DFS(G[v][i]);
    }
}
```

Можно модифицировать обход таким образом, чтобы хранить предков, или номера компонент связности.

Назовем **деревом обхода в глубину** подмножество из всех вершин и всех ребра, по которым мы прошли в новые вершины (с $color[G[v][i]] = 0$). Данное определение работает только для одной компоненты связности. Если компонент связности несколько, то можно ввсети понятие **леса обхода в глубину** аналогиным образом.

Все ребра после обхода в глубину можно разделить на типы: 1) Ребро дерева, вошедшее в дерево обхода в глубину. 2) Обратное - то, которое смотрит в предка. Также для ориентированных графов (ТОЛЬКО ДЛЯ НИХ): 3) Прямое - не вошедшее в дерево поиска, но смотрящее в предка 4) Перекрестное - все остальные ребра.

У каждой вершины в поиске в глубину можно ввести 2 параметра - время входа *tin* и время выхода *tout*. С помощью этих параметров можно легко классифицировать любое ребро.

```
vector<vector<int> > G;
vector<int> color;// color.assign(n, 0) in main
vector<int> tin;
vector<int> tout;
int t;
```

```
void DFS(int v)
{
    color[v] = 1;
    tin[v] = t;
    t++;
    for (int i = 0; i < G[v].size(); ++i)
    {
        if (color[G[v][i]] == 0)
            DFS(G[v][i]);
    }
    tout[v] = t;
    t++;
}
```

Топологическая сортировка Топологическая сортировка решает следующую задача. Дан ориентированный граф с n вершинами и m рёбрами. Требуется перенумеровать его вершины таким образом, чтобы каждое рёбро вело из вершины с меньшим номером в вершину с большим.

Иными словами, требуется найти перестановку вершин (топологический порядок), соответствующую порядку, задаваемому всеми рёбрами графа.

Топологическая сортировка может быть не единственной. Если в графе есть цикл, то топологической сортировки не существует.

топологическая сортировка — это сортировка в порядке убывания времён выхода *tout*.