Хранение графов

1 Введение

Есть 3 основных способа хранения графов:

1. матрица смежности,
2. список ребер,
3. список смежности.

Каждый из них лучше применим в тех или иных условиях. Рассмотрим для каждого из них чтение графа из входных данных (предполагается, что на вход граф подается в виде списка неориентированных ребер) и проход по всем ребрам для некоторой вершины.

2 Матрица смежности

Самой простой в написании вариант, однако и требует больше всего памяти, $O(N^2)$. Обычно применяется в случае графов, близких к плотным графам, т.к. количество ребер будет близко к квадрату количества вершин. Однако для разреженных графов метод можно применять в случае небольшого количества вершин. Уже при $N \approx 1000$ такая матрица будет расходовать большое количество памяти. Способ сам из себя представляет матрицу A размера $N \times N$, где A_{ij} показывает наличие ребра (i, j) в графе.

```
n, m = map(int, input().strip().split())
a = [[0] * n for _ in range(n)]
for _ in range(m):
    u, v = map(int, input().strip().split())
    a[u][v] = 1
    a[v][u] = 1

u  # we will iterate over all edges from u
for v in range(n):
    if a[u][v] == 1:
        # we have an edge (u, v)
```

Если распечатать матрицу смежности для неориентированного графа, то можно заметить, что она будет симметрична относительно главной диагонали. В таком случае при просмотре всех возможных ребер имеет смысл смотреть только верхний или нижний треугольник матрицы. Для взвешенных графов матрица будет хранить веса ребер.

3 Список ребер

Название метода тут говорит само за себя. Мы просто будем хранить массив пар вершин, между которыми проходит ребро. В случае неориентированного графа храним неориентированные ребра. Порядок хранения ребер не важен. Таким образом затрачивается памяти $O(M)$.

```
n, m = map(int, input().strip().split())
edges = []
for _ in range(m):
    u, v = map(int, input().strip().split())
    edges.append((u, v))

i  # we will iterate over all edges from i
for u, v in edges:
    if u == i or v == i
        # we have an edge (u == i, v) or (v == i, u)
```

Поменяем немного способ хранения ребер:

1. теперь мы храним ориентированные ребра,
2. список ребер отсортирован,
3. для каждой вершины храним позиции первого и последнего ребер в списке.

Таким образом нам не придется проходить по всему списку в поисках ребра для одной вершины. Однако такое улучшение требует $O(n+m)$ памяти.

```
n, m = map(int, input().strip().split())
edges = []
for _ in range(m):
    u, v = map(int, input().strip().split())
    edges.append((u, v))
    edges.append((v, u))
edges.sort()
positions = [(0, 0)] * n
prev = edges[0][0]
begin = 0
for i in range(1, m * 2):
    if edges[i][0] != prev:
        positions[prev] = (begin, i)
        prev = edges[i][0]
        begin = i
positions[prev] = (begin, len(edges))
```

```
u  # we will iterate over all edges from u
for _, v in edges[positions[u][0]:positions[u][1]]:
    # we have an edge (u, v)
```

4 Список смежности

Список смежности из себя представляет список списков или список сетов. Где каждый внутренний список/сет хранит вершины, в которые есть ребра из нашей вершины. Обратите внимание, что такой способ хранит только ориентированные ребра. Для работы с неориентированными графами придется хранить каждое неориентированное ребро как два ориентированных. Способ с сетами не работает для кратных ребер. Требования к памяти - $O(n + m)$.

```
n, m = map(int, input().strip().split())
adj_list = [set() for _ in range(n)]
for _ in range(m):
    u, v = map(int, input().strip().split())
    adj_list[u].add(v)
    adj_list[v].add(u)

u  # we will iterate over all edges from u
for v in adj_list[u]:
    # we have an edge (u, v)
```