Задача

Дан ациклический ориентированный граф. Найти за линейное время путь максимальной длины в этом графе.

Решение

Так как граф у нас ациклический, то такой путь в нём всегда будет. Если бы граф имел цикл, то путь максимальной длины был бы равен бесконечности.

Разобьём решение на две части:

1. найдём длины максимальных путей для каждой вершины
2. восстановим наидлиннейший путь

Граф будем хранить в виде списка смежнности.

1) Максимальные длины путей для каждой вершины

Данные длины можно найти используя динамическое программирования.

Для каждой вершины будем хранить максимальную длину пути, в который входит данная вершина.

Таким образом, чтобы найти максимально длинный путь для новой вершины, нужно выбрать максимально длинный путь из вершин соседей, в которую мы можем попасть. Если для неё ещё не выбран данный путь, то спускаемся в эту вершину и сначала ищем наидлиннейший путь для неё. Так пока не дойдём до вершины, из которой нет выхода, для неё наидлиннейший путь будет равен 1 (такая вершина обязательно будет, так как граф ациклический). Далее поднимаемся и высчитываем длины для остальных вершин.

Данный алгоритм очень хорошо описывается с помощью алгоритма *depth for search*.

```
def dfs(curr, G, visited, dp):
    if visited[curr]:
        return dp[curr]
    visited[curr] = True
    max_length = 0
    for vert in G[curr]:
        max_length = max(max_length, dfs(vert, G, visited, dp))
    dp[curr] = max_length + 1
    return dp[curr]
```

2) Восстановление наидлиннейшего пути

Для того, чтобы восстановить путь максимальной длины, найдём в массиве с длинами максимальную длину и индекс это вершины.

Сохраним эту веришну, а затем с этой вершины перейдём на вершину, у которой путь максимальной длины на 1 меньше (такая вершина обязательно будет, потому что максимальный путь для текущей вершины выбирается, как *максимальный путь для предыдущей + 1*).

Повторим данную операцию, пока текущий путь максимальной длины больше 0 (т.е. ещё существует).

```
ddef restore_path(G, dp):
    idx = max_length = -1
    for i, length in dp.items():
        if length > max_length:
            max_length = length
            idx = i

    path = []

    while max_length > 0:
        path.append(idx)
        max_length -= 1
        for vert in G[idx]:
            if dp[vert] == max_length:
                idx = vert
                break

    return path
```

Сложность

Алгоримт состоит из двух шагов:

1. *dfs* - его сложность `O(V + E)`, *V* - количество вершин, *E* - количество ребёр
2. восстановление пути максимальной длины - `O(V + E)`, *V* - количество вершин, *E* - количество ребёр

Итого: `O(E + V)` + `O(E + V)` = `O(E + V)`

Покажем, что сложность *dfs* и правда `O(E + V)`.

Граф хранится в виде списка смежности. В *dfs* мы перебираем все вершины (`O(V)`), а также все рёбра (`O(E)`). Тоесть всего выполняется `O(E)` + `O(V)` операций. Итоговая сложность `O(E + V)`.

В методе восстановления пути тот же *dfs*, но применён немного по-другому (итеративно спукаемся по вершинам). В худшем случае (граф представлен в виде односвязного списка) придётся перебрать все вершины и все рёбра. Это `O(E + V)`.

Code

```python
from collections import defaultdict


def dfs(curr, G, visited, dp):
    if visited[curr]:
        return dp[curr]
    visited[curr] = True
    max_length = 0
    for vert in G[curr]:
        max_length = max(max_length, dfs(vert, G, visited, dp))
    dp[curr] = max_length + 1
    return dp[curr]


def restore_path(G, dp):
    idx = max_length = -1
    for i, length in dp.items():
        if length > max_length:
            max_length = length
            idx = i

    path = []

    while max_length > 0:
        path.append(idx)
        max_length -= 1
        for vert in G[idx]:
            if dp[vert] == max_length:
                idx = vert
                break

    return path


def longest_path(G):
    visited = {vert: False for vert in G}
    dp = defaultdict(int)
    for vert in G:
        if not visited[vert]:
            dfs(vert, G, visited, dp)

    return restore_path(G, dp)
```