Контрольные вопросы

6 марта 2025 г.

1) Граф - это набор точек (вершин) и ребер (отрезков, соединящих две вершины). Обозначаются как $G(E, V)$

2) Наибольшее количество ребер M в неориентрованном графе - $\frac{N \cdot (N-1)}{2}$, так как каждая вершина не имеет связи с собой и имеет максимум одно ребро с каждой из вершин. В случае ориентированного графа ребро может соединять вершины в одном направлении, но не в обратном. Поэтому в случае ориентрованного графа N $\leq N(N-1)$

3) Проверить направленность графа можно при помощи **матрицы смежности**. Если она будет симметрична-граф неориентированный .То есть если поменять строку и столбец местами (поменять местами две вершины), значение в матрице должно совпадать.

4) При добавлении веса каждому ребру в списке ребер вместо пары чисел должна вводится тройка. В списке смежностей помимо каждой вершины должен вводится вес ребра, ведущего к этой вершине.

5) Компонента связности- это такая часть графа, между любыми двумя вершинами которой есть связь (через одно ребро либо через несколько вершин, входящих в компоненту).

6) При помощи BFS можно искать циклы в графе. Если нужно найти кратчайший цикл, то выгоднее будет использовать BFS, в случае, если нужно найти любой цикл, будет выгоднее использовать DFS.

7) Алгоритм Дейкстра не работает с отрицательными ребрами, так как не будет учитываться возможность найти более короткий путь к вершине. Решить это можно прибавив какое-нибудь большое число к отрицательным ребрам и вычесть его из итоговой длины столько раз, сколько отрицательных ребер встретилось на пути к этой вершине.

8) Алгоритм Форд Беллмана работает N-1 раз, чтобы найти проверить все пути и запускается N-й раз для проверки отриуательных циклов, если расстояние меняется, то в этом случае искать кратчайшее расстояние будет некорректно, так как есть отрицательный цикл, прроходя по коттрому расстояние будет уменьшаться сколько угодно раз

9) Алгоритм Форд-Беллмана будет работать быстрее только в случае маленького числа M или сильно разреженного графа. Тогда асимптотика будет $O(N)$ и так как операци более простые, константа будет меньше.

10) В случае, если глубина рекурсии будетт слишком большая, стек может быть переполнен и будет выгоднее использовать итеративную реализацию DFS. В случае не слишком большой глубины выгоднее использовать рекурсию. Использование списка как стека будет эффективно, так как добавление и удаление элемента происходит за O(1). В случае BFS использовать очередь будет неэффективно так как удаление из начала очереди будет происходить за O(1), поэтому лучше использовать стек.

1 Задания

1.

```
def edge_list():
    N = int(input())
    M = int(input())
    res = []
    for i in range(M):
        x, y = (int(x) for x in input().split())
        res.append((x, y))
    return N, res
```

2.

```
def adj_matrix(N, edge_list):
    res = [[0 for j in range(N)] for i in range(N)]

    for x, y in edge_list:
        res[x][y] = 1
    return res
```

3.

```
def reversed_adj(matrix):
    N = len(matrix)
    reversed_matrix = [[0] * N for _ in range(N)]
    for i in range(N):
        for j in range(N):
            reversed_matrix[j][i] = matrix[i][j]

    return reversed_matrix
```

4.

```
def components(adj_list):
    def BFS(start_node, visited):
        queue = [start_node]
        visited.add(start_node)
        while queue:
            curr_node = queue.pop(0)
            for adj_node in adj_list.get(curr_node, []):
                if adj_node not in visited:
                    visited.add(adj_node)
                    queue.append(adj_node)
    visited = set()
    count = 0
    for node in adj_list:
        if node not in visited:
            BFS(node, visited)
            count += 1
    return count
```

5.

```
def DFS(curr_node, adj_list, visited=None):
    if visited is None:
        visited = set()

    print(curr_node)
    visited.add(curr_node)

    for adj_node in adj_list[curr_node]:
        if adj_node not in visited:
            DFS(adj_node, adj_list, visited)
    print(curr_node)
    return visited
```

6.

```
def FB(curr_node, adj_list):
    d = [float("inf")] * len(adj_list)
    d[curr_node] = 0
    for i in range(len(adj_list) - 1):
        for x in range(len(adj_list)):
            for y in adj_list[x]:
                if d[x] != float("inf") and d[x] + 1 < d[y]:
                    d[y] = d[x] + 1
    for x in range(len(adj_list)):
        for y in adj_list[x]:
            if d[x] != float("inf") and d[x] + 1 < d[y]:
                print("Отрицательный цикл")

    return d
```