ЛЕКЦИЯ 20
Задачи на графы

1. Поиск минимального остовного дерева

Минимальное остовное дерево (для связного графа) — остовное дерево минимального суммарного веса.
Остовных деревьев может быть несколько. Есть 2 алгоритма поиска минимального остовного дерева: Прима и Краскала. Оба алгоритмы жадные. В первом случае мы выхватываем ребро, которое короче. Связность появится в последний момент. Во втором случае мы постоянно поддерживаем связанность. Условие: связный неориентированный граф.

1.1. Алгоритм Краскала

1.1.1. Алгоритм

1. Упорядочиваем все ребра по возрастанию веса (не убыванию).
2. Далее добавляем ребро по порядку к каркасу (вершины добавляются вместе с ребром), если оно не образует цикл с уже имеющимися ранее ребрами.

Подробная визуализация разобрана на википедии.

1.1.2. Реализация

Программа №1.1. Реализация алгоритма Краскала

```
1    N, M = [int(x) for x in input().split()]
2    edges = []
3    for i in range(M):
4        v1, v2, weight = map(int, input().split())
5        edges.append((weight, v1, v2)) # Сначала будем добавлять вес
6    edges.sort()
7    comp = list(range(N))
8    tree = []
9    tree_weight = 0
10   for weight, v1, v2 in edges:
11       if comp[v1] != comp[v2]:
12           tree.append((v1, v2))
13           tree_weight += weight
14           for i in range(N):
15               if comp[i] == comp[v2]:
16                   comp[i] = comp[v1]
```

Сложность алгоритма $O(M \log M + M \cdot N)$.
Построчный комментарий кода:
1) Вводим количество вершин и ребер.
2) Создаем список для ребер.
4) Вводим ребра — вершины, которые они соединяют и вес.

5) Добавляем в список ребер.
6) Сортируем по весу ребер.
7) Создаем список ребер, чтобы работать с индексами и проверять, из одной они компоненты связанности или нет.
8) Сам массив ребер, образующих остовное дерево.
9) Вес графа.
10) Пока не прошли по всем ребрам.
11) Если вершины не принадлежат одной компоненте связанности.
12) Добавляем ребро к остовному дереву.
13) Добавляем в общий вес вес нового ребра.
14) Проходим по всем вершинам.
15-16) Если мы встречаем аналогичную второй вершине компоненту связанности,то перекрашиваем ее в компоненту связанности первой. Так свяжем дерево.

1.2. Алгоритм Прима

1.2.1. Алгоритм

1. Выбираем произвольную вершину (она и есть заготовка для дерева).
2. Добавляем к каркасу ребро минимального веса среди ребер, инцидентных какой-либо вершине каркаса и вершине не из каркаса.

1.2.2. Реализация

`dist[i]` — минимальный вес ребра, которым можно присоединить вершину i к каркасу.

Программа №1.2. Реализация

```
1    INF = 10**9 # Введем условную бесконечность
2    dist = [INF]*N # W[i][j] - вес ребра ij, который равен +бесконечность,
     если i не смежна j
3    dist[0] = 0
4    used = [False]*N
5    used[0] = True
6    tree = []
7    tree_weight = 0
8    for i in range(N):
9        min_d = INF
10       for j in range(N):
11           if not used[j] and dist[j] < min_d:
12               min_d = dist[j]
13               u = j
14       tree.append((i, u))
15       tree_weight += min_d
16       used[u] = True
17       for v in range(N):
18           dist[v] = min(dist[v], W[u][v])
```

Асимптотика алгоритма: $O(N^2)$. С помощью кучи можно ускорить до $O((M+N)\log N)$.

2. Алгоритм построения Гамильтонова цикла

Гамильтонов цикл — цикл, проходящий через все вершины по одному разу. Гамильтонов путь — путь, проходящий через все вершины по одному разу.
Это NP – сложная задача, т.к. решение задачи находится перебором, $O(N!)$.

Программа №2.1. Реализация

```
1    visited = [False]*N
2    path = []
3    def hamilton(curr):
4        path.append(curr)
5        if len(path) == N:
6            if A[path[-1]][path[0]] == 1: # A - таблица смежности
7                return True
8            else:
9                path.pop()
10               return False
11       visited[curr] = True
12       for next in range(N):
13           if A[curr][next] == 1 and not visited[next]:
14               if hamilton(next):
15                   return True
16       visited[curr] = False
17       path.pop()
18       return False
```

Г. С. Демьянов, VK
С. С. Клявинек, VK