Задача

Модифицировать алгоритм Прима из лекции так, чтобы он имел сложность не хуже, чем:

1. O(E log V)
2. O(E + V log V)

Решение

В общем алгоритм Прима для построения минимального остовного дерева заключается в следующем:

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Важно: граф должен быть связным и неориентированным.

1) Решение за *O(E log(V))*

Здесь для выбора ребра с наименьшей стоимостью будем использовать *Binary Heap*.

Используем алгоритм из лекции с ленивым удалением.

Сложность

Сложность такого алгоритма, как было доказано, *O(E log V)*, потому что *E* раз происходит добавление и удаление в случае такого алгоритма.

Code

```
def binary_heap_mst(graph):
    heap = []
    mst = set()
    edges = []
    tree_weight = 0
    next_vert = next(iter(graph.keys()))
    mst.add(next_vert)
    while len(mst) < len(graph):
        v = next_vert
        for vert, weight in graph[v]:
            heapq.heappush(heap, (weight, v, vert))

        while heap[0][2] in mst:
            heapq.heappop(heap)

        weight, v1, v2 = heap[0]
        heapq.heappop(heap)
        next_vert = v2
        mst.add(v2)
        edges.append((v1, v2))
        tree_weight += weight

    return edges, tree_weight
```

2) Решение за *O(E + V log V)*

Здесь для выбора ребра с наименьшей стоимостью будем использовать *Fibonacci Heap*.

В алгоритме сначала потребуется инициализировать все вершины с весом равным бесконечности (очень большим, чтобы был больше веса любого из ребёр). Сложность данной операции *O(V)*.

Далее начнём алгоритм с какой-то вершины и для каждого ребра данной вершины, изменим вес соседней вершины, соединённой с данной этим рёбром, и вершину, и вторую вершину, если текущий вес меньше хранимого веса в данной вершине (данная операция выполняется за *O(1)*. Далее извлекаем минимальный элемент из кучи (*O(log V)*) и добавляем вершину в остовное дерево. После рассматриваем добавленную вершину как текущую и повторяем операцию пока все вершины не будут в остовном дереве.

Всего мы рассмотрим *E* рёбер с операцией обновления за *O(1)* и *V* операций удаления минимума из кучи за *O(log V)*.

Сложность

Сложность такого алгоритма будет *O(V)* - на инициализацию + *O(E * 1)* - на обновление веса ребёр + *O(V * log V)* - на извлечения минимума из кучи.

Итого: *O(V + E + V * log V) = O(E + V * log V)*.

Code

```
def fibonacci_heap_mst(graph):
    fheap = FibonacciHeap()
    nodes = {}
    mst_weight = 0
    edges = []
    mst = set()
    for v in graph:
        nodes[v] = fheap.insert((float('inf'), -1, -1))

    next_vert = next(iter(graph.keys()))
    mst.add(next_vert)
    while len(mst) < len(graph):
        v = next_vert
        for vert, weight in graph[v]:
            if vert not in mst:
                fheap.decrease_key(nodes[vert], (weight, v, vert))

        weight, v1, v2 = fheap.extract_min().key
        next_vert = v2
        mst.add(v2)
        edges.append((v1, v2))
        mst_weight += weight

    return edges, mst_weight
```