۹۹۳۱۰۴۱ محمدحسین تقوی

(۱)

```
def floyd_warshall(adj_matrix):
    n = len(adj_matrix)
    for i in range(n):
        for j in range(n):
            if i != j and adj_matrix == 0:
                adj_matrix ← +∞
    for i in range(n):
        for j in rang(n):
            for k in range(n):
                if adj_matrix[j][i] >
                    adj_matrix[j][i] + adj_matrix[i][k]:
                adj_matrix[j][k] = adj_matrix[j][i] +
                    adj_matrix[i][k]
    return adj_matrix
```

MEHR

(۲) کدی است از الگوریتم کروسکال استفاده کنیم

ابتدا یال ها را براساس وزنشان به صورت صعودی

مرتب می کنیم که از مرتبه $O(E \log E)$ زمان می برد.

سپس در هر مرحله یال را به درخت اضافه می کنیم و چک می کنیم که

تشکیل دایره ندهد که این کار را با Union_Find انجام می دهیم که

خود ابتدا حداکثر به تعداد رأس ها مکروه دارد که ساختن

Union_Find از مرتبه $O(V)$ است.

حال برای چک کردن شرط حلقه و همچنین نیاز merge کردن مجموعه ها

برای هر یال به $O(V)$ بررسی نیاز است که در کل به

$O(E \log V)$ بررسی نیاز است

در کل هم به $O(E \log E + V + E \log V)$

نیاز است که اگر $E > V$ آنگاه $O(E \log V)$

و بالعکس $O(E \log E)$ نیاز است.

```
def kruskal(V, E):
    MST = []
    edges = sort(E)
    Union_Find(V)
    for edge in edges:
        if find(edge.u) != find(edge.v):
            union(edge.u, edge.v)
            MST.append(edge)
            if MST.length == len(V) - 1:
                break
    return MST
```

```
def Union_Find(V):
    parent = []
    rank = []
    for v in V:
        parent[v] = v
        rank[v] = 0

def find(u):
    if parent[u] != u:
        parent[u] = find(parent[u])
    return parent[u]
```

```
def union(u, v):
    root_u = find(u)
    root_v = find(v)
    if root_u != root_v:
        if rank[root_u] > rank[root_v]:
            parent[root_v] = root_u
        elif rank[root_u] < rank[root_v]:
            parent[root_u] = root_v
        else:
            parent[root_v] = root_u
            rank[root_u] ++
```