```
G1 = {
  0: {1:1, 2:3, 3:4},
  1: {0:1, 2:5},
  2: {0:3, 1:5, 3:2},
  3: {2:2, 0:4}
}

G2 = {
  'a': {'b':1, 'f':2},
  'b': {'a':1, 'c':1},
  'c': {'b':1},
  'd': {'f':1, 'e':2},
  'e': {'d':2, 'g':1},
  'f': {'a':2, 'd':1},
  'g': {'e':1}
}


def find(C, u):
    if C[u] != u:
        C[u] = find(C, C[u])                    # Path compression
    return C[u]

def union(C, R, u, v):
    u, v = find(C, u), find(C, v)
    if R[u] > R[v]:                             # Union by rank
        C[v] = u
    else:
        C[u] = v
    if R[u] == R[v]:                            # A tie: Move v up a level
        R[v] += 1

def kruskal(G):
    E = [(G[u][v],u,v) for u in G for v in G[u]]
    T = set()
    C, R = {u:u for u in G}, {u:0 for u in G}    # Comp. reps and ranks
    for _, u, v in sorted(E):
        if find(C, u) != find(C, v):
            T.add((u, v))
            print (u, v)
            union(C, R, u, v)
    return T

print list(kruskal(G2))
```

```
[('d', 'e'), ('e', 'g'), ('d', 'f'), ('b', 'c'), ('a', 'f'), ('a', 'b')]
```

Sedgewick, R. *Algorithms*, sf. 409

Heatland, Python Algorithms