Kruskal's Algorithm

这应该是一个贪心算法，我们维护一个以权排序的优先队列，每次取出权最小的边，并把两个顶点连起来。为了防止成环，我们使用disjoint set，每次连接操作的时候对两个顶点进行union操作，然后在连接前要判断是不是已经在一个component里了，若是，则跳过这条边

伪代码：

```
KruskalMST(G):
  DisjointSets forest
  foreach (Vertex v: G):
    forest.makeSet(v)

  PriorityQueue Q  // min edge weight
  foreach (Edge e : G):
    Q.insert(e)

  while |T.edges()| < n-1:
    Edge (u, v) = Q.removeMin()
    if forest.find(u) != forest.find(v):
      T.addEdge(u, v)
      forest.union(forest.find(u),
                   forest.find(v))

  return T
```

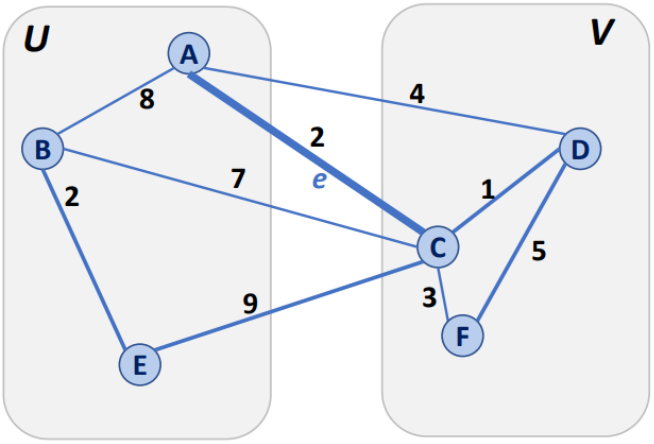
时间复杂度

不论优先队列是用堆还是有序数组实现的，最终的时间复杂度都是$O(m \log m)$

Prim's Algorithm

Partition Property

对于一个图，我们考虑一个任意的分割，分成两部分



让e是两个partition之间边权最小的边，那么e是最小生成树的一部分

Prim's Algorithm

伪代码

```
PrimMST(G, s):
  Input: G, Graph;
         s, vertex in G, starting vertex
  Output: T, a minimum spanning tree (MST) of G

  foreach (Vertex v : G):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  PriorityQueue Q  // min distance, defined by d[v]
  Q.buildHeap(G.vertices())
  Graph T          // "labeled set"
```

```
14
15    repeat n times:
16      Vertex m = Q.removeMin()
17      T.add(m)
18      foreach (Vertex v : neightbors of m not in T):
19        if cost(v, m) < d[v]:
20          d[v] = cost(v, m)
21          p[v] = m
22
23    return T
```

运行时间分析